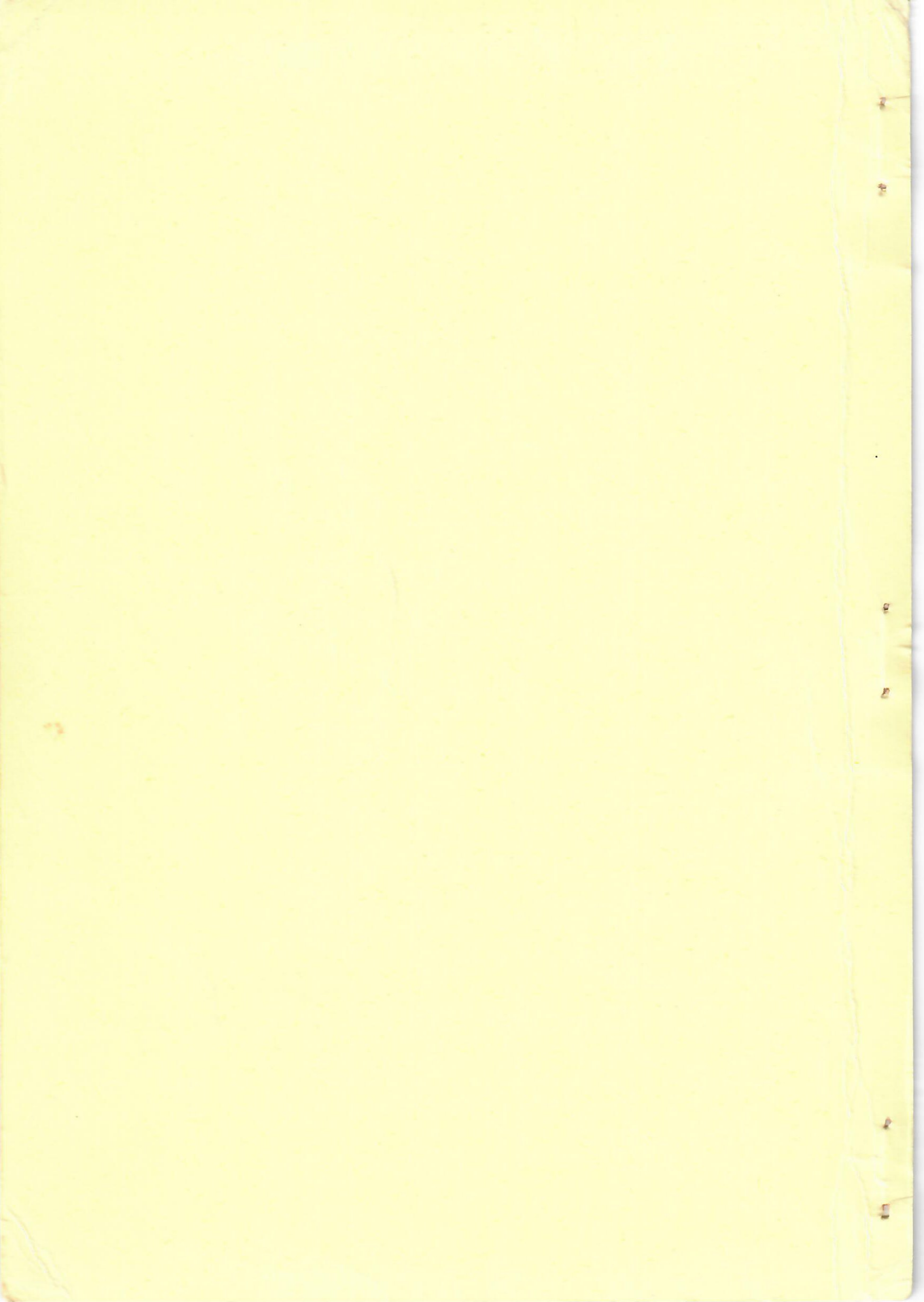
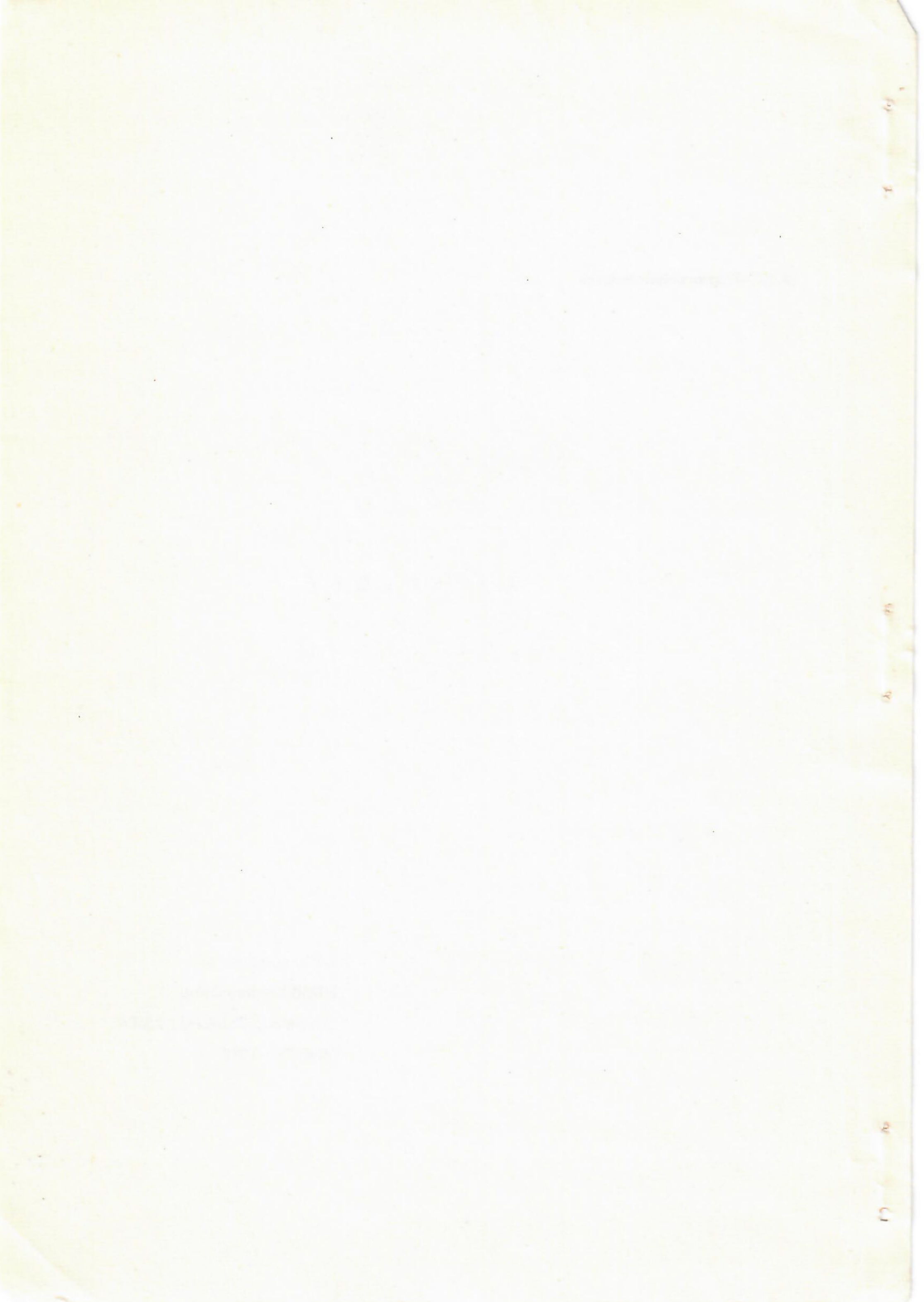
BASIC
Programmierhandbuch
Applesoft BASIC



BASIC-Programmierhandbuch

A P P L E S O F T

apple computer inc.
10260 Bandley Drive
Cupertino, California 95014
(408) 996-1010



Find'st Fehler Du in diesem Buch,
dann nicht den Korrektoren fluch'
Wär' Argus selbst dabeigewesen,
auch er hätt' manches überlesen.

Richard Shacklock, 1565
(Nachdichtung)

Herausgeber:

APPLE COMPUTER INC.
1Ø26Ø Bandley Drive
Cupertino, California 95Ø14
(4Ø8) 996-1Ø1Ø

Nachbestellungen: APPLE Product Nr. A2LØØØ6
(Ø3Ø-ØØ13-Ø3)

APPLESOFT II - ERWEITERTES BASIC MIT GLEITKOMMA

Kurze Übersicht

Einfache Variable

Art	Bezeichnung	Bereich
reel	AB	+/- 9.99999999 E+37
ganz	AB %	+/- 32767
Datenfolge	AB$	Ø bis 255 Zeichen

Wo A ein Buchstabe ist, ist B ein Buchstabe oder eine Ziffer.

Die Bezeichnung kann aus mehr als zwei Zeichen bestehen, aber nur die ersten beiden sind von Bedeutung. So sind AB% und AB3QS% die gleichen ganzen Variablen.

Bereichsvariable

Art	Bezeichnung des typischen Elements
real	AB(3,12,7)
ganz	AB%(3,12,7)
Datenfolge	AB$(3,12,7)

Die Bereichsgröße wird durch den verfügbaren Speicher begrenzt.

Algebraische Operatoren

=	ordnet der Variable einen Wert zu (wahlweise LET)
-	Negation
^	Potenzieren
*	Multiplizieren
/	Dividieren
+	Addieren
-	Subtrahieren

Vergleichs- und Logikoperatoren

=	bedeutet "gleich"
<>	bedeutet "nicht gleich"
>	bedeutet "größer als"
<	bedeutet "kleiner als"
<=	bedeutet "kleiner als oder gleich"
>=	bedeutet "größer als oder gleich"

NOT	bedeutet logisches "nicht"
AND	bedeutet logisches "und"
OR	bedeutet logisches "oder"

Vergleichende und logische Ausdrücke besitzen den Wert 1, wenn sie wahr sind, den Wert Ø, wenn sie falsch sind. Der Vergleichsoperator kann auch zum Vergleich von Datenfolgen (Strings) verwendet werden.

System- und Programmbefehle

LOAD	Ladet ein Programm vom Band.
SAVE	Das Programm wird auf Band aufgezeichnet.
NEW	Damit wird das laufende Programm gelöscht.
RUN	Das Programm läuft bei der niedrigsten Zeilennummer beginnend ab.
RUN 477	Das Programm läuft bei Zeilennummer 477 beginnend ab.
STOP	Unterbricht den Programmablauf und zeigt die Zeilennummer der Unterbrechung an.
END	Beendet die Ausführung des Programms ohne Anzeige einer Nachricht.
ctrl C	Wird zur unmittelbaren Unterbrechung des Programms oder des Auflistens benutzt, Anzeige Zeilennr.
reset	Der Druck der Taste führt zu einem unbedingten Sprung zum Systemmonitor. Verwende ctrl C oder ØG, um zu Applesoft zurückzukehren.

CONT	Ein durch STOP, END oder ctrl C unterbrochenes Programm wird fortgesetzt.
TRACE	Fehlersuchhilfe, listet jede Zeilennummer nach der Ausführung auf.
NOTRACE	Abstellung der Fehlersuche.
PEEK(X)	Wiedergabe des Inhalts von Speicherstelle X.
POKE X,13	Verändert den Inhalt der Speicherstelle X zum Wert 13.
WAIT X,Y,Z	Unterbricht solange, bis der Inhalt von Stelle X ein von Null verschiedenes Resultat ergibt, nachdem X mit Z über "oder" und mit Y über "und" verglichen wurde.
CALL X	Springt zum maschinensprachlichen Unterprogramm, beginnend bei Speicherstelle X.
USR(X)	Überträgt den Wert X in ein maschinensprachliches Unterprogramm.
HIMEM:	Stellt die höchstmögliche Adresse im Speicher ein, die für das APPLESOFT-Programm zur Verfügung steht.
LOMEM:	Stellt die niedrigste Adresse im Speicher ein, die für das APPLESOFT-Programm zur Verfügung steht.

Schreibbefehle und Befehle zum Format

LIST	Listet das ganze Programm auf.
LIST X-Y	Listet von Zeile X bis Zeile Y auf.
DEL X,Y	Löscht von Zeile X bis Zeile Y.
REM XYZ	Zum Schreiben von Programmbemerkungen, die nicht vom Programm verarbeitet werden.
VTAB Y	Bewegt den Cursor auf Spalte Y (1 bis 24) - vertikal.
HTAB X	Bewegt den Cursor zur Position X (1 bis 4Ø) - horizontal.
TAB(X)	Gilt nur für PRINT-Anweisung, bewegt den Cursor auf Position X (1 bis 4Ø).

POS(Ø)	Stellt den gegenwärtigen horizontalen Standort des Cursors auf (Ø bis 39) zurück.
SPC(X)	Gilt nur für PRINT-Anweisung, läßt X Leerstellen zwischen dem zuletzt gedruckten und dem als nächstes zu druckende Zeichen.
HOME	Löscht den Bildschirm und bringt den Cursor in die Stellung oben links.
CLEAR	Alle Variablen werden auf Null gesetzt.
FRE(Ø)	Gibt dem Benutzer die noch zur Verfügung stehende Speicherkapazität an.
FLASH	Läßt die Bildschirmangabe blinken.
INVERSE	Läßt die Bildschirmanzeige schwarz auf weißem Grund erscheinen.
NORMAL	Befehle FLASH oder INVERSE werden rückgängig gemacht.
SPEED=X	Einstellen der Zeichen Output-Rate (Ø bis 255).
esc A	Bewegt den Cursor um eine Stelle nach rechts.
esc B	Bewegt den Cursor um eine Stelle nach links.
esc C	Bewegt den Cursor um eine Stelle nach unten.
esc D	Bewegt den Cursor um eine Stelle nach oben.
Pfeil nach links	Löscht das Zeichen der Zeile, das gerade gedruckt wurde und bewegt den Cursor um eine Stelle nach links.
Pfeil nach rechts	Gibt Zeichen unter dem Cursor in den Speicher ein und bewegt den Cursor um einen Abstand nach rechts.
ctrl X	Löscht die gerade geschriebene Zeile.

Bereiche und Datenfolgen

DIM A (X,Y,Z)	Eintasten von maximalen Indices für A. Speicherraum für reale Elemente X+1 * Y+1 * Z+1, beginnend mit A (Ø,Ø,Ø), bleibt frei.

DIM A$ (X,Y)	Eintasten von maximalen Indices für A$, die X+1 [illegible] Y+1 Datenfolgenelemente enthalten können, wobei ein jedes aus bis zu 225 Zeichen bestehen kann.
LEN (A$)	Gibt die Zahl der Zeichen in A$ wieder
STR $ (X)	Gibt den Zahlenwert von X wieder, umgewandelt in eine Datenfolge
VAL (A$)	Gibt A$ bis zum ersten nicht numerischen Zeichen als einen Zahlenwert wieder.
CHR $ (X)	Gibt das ASC II-Zeichen wieder, dessen Code X ist.
ASC (A$)	Gibt den ASC II-Code für das erste Zeichen der A$ wieder.
LEFT $ (A$,X)	Gibt X Zeichen von A$ wieder, vom linken Ende an gerechnet.
RIGHT $ (A$,X)	Gibt X Zeichen von A$ wieder, vom rechten Ende an gerechnet.
MID$ (A$,X,Y)	Gibt Y Zeichen von A$, beginnend bei Zeichen X, wieder.
+	Operator zum Verketten von Datenfolgen.
STORE A	Zeichnet die Zahlenreihe A auf Band auf. Kann nicht benutzt werden, Datenfolgenbereiche direkt aufzuzeichnen.
RECALL B	Ruft den Bereich vom Band ab. Bereich B muß richtig geDIMed worden sein, siehe Anweisung DIM!

Input/Output-Befehle

(Siehe auch LOAD und SAVE sowie STORE und RECALL)

INPUT A$	Läßt ein "?" auf dem Bildschirm erscheinen. Computer ist bereit, einen vom Benutzer eingetasteten Datenfolgenwert für A$ entgegenzunehmen.
INPUT "XYZ";A	XYZ erscheint auf dem Bildschirm. Computer ist bereit, einen vom Nutzer für A eingegebenen reellen Zahlenwert entgegenzunehmen.
GET A$	Computer ist bereit, einen aus einem Zeichen bestehenden Wert für A$ anzunehmen. RETURN-Taste braucht nicht betätigt zu werden.

DATA X,"Y",Z	Eine Liste von Datenelementen wird aufgestellt, die für eine READ-Anweisung genutzt werden kann.
READ A$	Weist A$ das nächste DATA-Element zu.
RESTORE	Führt das READ noch einmal aus, beginnend mit dem ersten DATA-Element.
PRINT "X=";X	Macht die Datenfolge X= und den Wert der Variable X auf dem Bildschirm sichtbar. Semikolons verketten die gedruckten Folgen miteinander, Kommas reihen die Posten in drei Tabellenbereiche ein. Das Symbol "?" heißt ebenfalls PRINT.
IN#6	Operator gewährleistet, daß künftig INPUTS über den Peripheriestecker Nr. 6 erfolgen und nicht über das Tastenfeld (IN#Ø).
PR#6	Output wird über Peripheriestecker Nr. 6 auf ein externes Gerät überspielt, und nicht auf den Bildschirm (PR#Ø).
LET X=Y	LET ist wahlweise. Es ordnet den Wert Y der Variablen X zu.
DEF FN A(X)=X+23/X	Definiert eine Funktion A. Im Späteren Gebrauch wird das Argument der Funktion A ersetzt durch den definierten Ausdruck von X. FNA (4) würde 9,75 ergeben.

<u>Befehle zur Steuerung des Programmablaufs</u>

GOTO 347	Verzweigt auf Zeile 347.
IF X=3 THEN STOP	Wenn die Behauptung X=3 wahr ist (ungleich Null), dann läuft das Programm weiter. Ist die Behauptung falsch, dann kommt es zu einem Sprung zur nächsten numerierten Zeile.
FOR X=1 TO 2Ø STEP 4...NEXT X	Es werden alle Anweisungen zwischen der FOR-Anweisung und der entsprechenden NEXT-Anweisung ausgeführt, zuerst mit X=1, dann mit X=5, X=9 usw. bis X>2Ø, wenn sich die Programmausführung nach der NEXT-Anweisung fortsetzt. STEP-Maß ist 1, falls nicht anders festgelegt.

NEXT X	Definiert die Basis der FOR...NEXT-Schleife. X ist wahlweise.
GOSUB 33	Verzweigt zu dem Unterprogramm bei Zeile 33.
RETURN	Kennzeichnet das Ende des Unterprogramms. Geht auf die Anweisung, die dem letzten GOSUB folgt, zurück.
POP	Entfernt eine Adresse vom RETURN-Adressenstapel.
ON X GOTO 397,12,458	Verursacht die Verzweigung zur x-ten Zeilennummer in der Liste. Wenn X=2, geht der Computer auf Linie 12.
ON X GOSUB 397,12,458	Löst die Verzweigung zum Unterprogramm bei der x-ten Zeilennummer in der Liste aus.
ONERR GOTO 45ØØ	Nachfolgende Fehler verursachten eine Verzweigung zur Zeile 45ØØ, in der eine Routine zur Fehlerbehandlung beginnt, ohne daß die Fehlernachricht gegeben wird und die Programmausführung angehalten wird.
RESUME	In der Fehlerbehandlungsroutine wird zu der Anweisung zurückgegangen, wo der Fehler aufgetreten ist.

Graphische Darstellungen und Spielsteuerung

Graphische Darstellungen mit niedriger Auflösung

GR	Stellt die graphische Darstellung mit niedriger Auflösung ein. Oben wird ein Leerraum von 4Ø x 4Ø Punkten geschaffen. Unten bleiben vier Zeilen für Text.
COLOR=X	Stellt die Farbe für die als nächstes darzustellende Abbildung ein. Farbskala (Ø bis 15).
PLOT X,Y	Stellt den farbigen Punkt auf die Koordinaten X (horizontal) und y (vertikal) ein. Koordinatenbereich von x und y erstreckt sich von Ø bis 39. (Ø,Ø) befindet sich ganz oben links.
HLIN X1,X2 AT Y	Eine horizontale Linie wird gezogen zwischen den Punkten X1,Y und X2,Y.

VLIN Y1, Y2 AT X	Zwischen den Punkten X, Y1 und X, Y2 wird eine vertikale Verbindung hergestellt.
SCRN (X, Y)	Gibt die eingestellte Farbe bei Punkt X, Y auf dem Bildschirm wieder (Skalenwert).

Graphische Darstellungen mit hoher Auflösung

HGR	Stellt auf graphische Darstellung mit hoher Auflösung, Seite 1, ein. Dabei wird oben ein Freiraum von 280 x 160 Punkten geschaffen. Unten bleiben vier Zeilen für Text.
HGR2	Stellt auf graphische Darstellung mit hoher Auflösung, Seite 2, ein. Dabei wird der ganze 280 x 192 große Raum freigemacht.
HCOLOR=X	Stellt die Farbe (0 bis 7) für die nächste Abbildung ein.
HPLOT X, Y	Stellt den farbigen Punkt auf die Koordinaten x (horizontal) und y (vertikal) ein. Der Bereich von x erstreckt sich von 0 bis 279, und der von y erstreckt sich von 0 bis 159 (bei HGR) oder bis 191 (bei HGR 2). Der Punkt (Ø, Ø) befindet sich in der linken oberen Ecke.
HPLOT X1, Y1 TO X2, Y2	Zwischen den Punkten (X1, Y1) und (X2, Y2) wird eine Linie gezogen. Der Befehl läßt sich auf weitere Punkte, TO XN, YN erweitern.
SHLOAD	Dadurch wird eine Formtabelle vom Band geladen.
DRAW 3 AT X, Y	Zeichnet die Formdefination #3 entsprechend der vorher eingegebenen Formtabelle, beginnend mit Punkt X, Y in der Farbe, die durch HCOLOR festgelegt wurde.
XDRAW 3 AT X, Y	Zeichnet die Formdefination #3 nach der Formtabelle, wobei jeder Punkt in der Komplementärfarbe des Punktes abgebildet wird, die dort auf dem Bildschirm ursprünglich gesetzt wurde.
ROT=X	Bewirkt die Rotation der Form für den DRAW oder den XDRAW-Befehl. ROT=0 entspricht "vertikal", ROT=16 bedeutet 90° in Uhrzeigersinn, ROT=32 entspricht 180° usw.

SCALE=X — Stellt die Skala (1 bis 255) der Form für den DRAW oder den XDRAW-Befehl ein.

Spielsteuerung

PDL (X) — Macht die Einstellung von Ø bis 255 der Spielsteuerung X (Ø bis 3) rückgängig.

PEEK (X-16287) — Wenn > 127, wird der Knopf auf Spielsteuerung X (Ø bis 2) gedrückt.

PEEK(-16336) — "Click" ertönt über den Lautsprecher des Computers.

Einige mathematische Funktionen

SIN(X) — Gibt den Sinus eines Winkels X an (Bogenmaß).

COS(X) — Gibt denCosinus eines Winkels X an (Bogenmaß).

TAN(X) — Gibt den Tangens eines Winkels X an (Bogenmaß).

ATN(X) — Gibt den Wert von Arcustangens von X in Bogenmaß an.

INT(X) — Gibt die größte ganze Zahl an, die ≤ X ist.

RND(1) — Gibt jedesmal eine zufällige reelle Zahl von Ø bis Ø,999999999, wenn diese benötigt wird, an.

RND(Ø) — Gibt die letzte zufällige Zahl wieder.

RND (-3) — Gibt 4,48217179E-Ø8 wieder; eine unterschiedliche feststehende Zahl wird für jedes unterschiedliche negative Argument wiedergegeben. Danach wird RND mit positivem Argument einer festgelegten Sequenz folgen.

SGN(X) — Gibt -1 wieder, wenn X < Ø,Ø, wenn X=Ø und 1, wenn X > Ø.

ABS(X) — Gibt den absoluten Wert von X wieder.

SQR(X) — Gibt die positive Quadratwurzel von X wieder.

EXP(X) — Gibt e^X wieder. (e = 2,718289)

LOG(X) — Gibt den natürlichen Logarithmus von X wieder.

INHALTSVERZEICHNIS

ÜBERSICHT

KAPITEL 1

KAPITEL 2

Definitionen

KAPITEL 3

Editieren, formatbezogene
Redigieren und Formatangaben Anweisungen

KAPITEL 4

Vektoren & Matrizen

Felder und Zeichenfolgen

KAPITEL 5

Bereiche und Gruppen

Ein-/Ausgabe-

KAPITEL 6

Input/Output-Befehle

Siehe auch Kapitel 3 unter LOAD und SAVE sowie Kapitel 5 unter STORE und RECALL.

KAPITEL 7

Befehle zum Steuerungsablauf

KAPITEL 8

Graphische Darstellungen und Spielsteuerung — Seite

Frei definierbare Zeichen (Symbole)
der Einzelpunkt-Graphik (

KAPITEL 9

Formative für Darstellungen im HR-Betrieb

Seite

freidef. Zeich. (Symb.)

KAPITEL 10

ANLAGEN

ÜBERSICHT

Einleitung

Bei APPLESOFT-II-BASIC handelt es sich um die von APPLE stark erweiterte BASIC-Sprache. BASIC wurde erweitert, weil unser Erzeugnis, der Computer APPLE II, so viele Merkmale hat, die bei anderen Computern, die lediglich mit BASIC arbeiten, einfach nicht vorhanden sind. Durch das Hinzufügen einiger weniger neuer Worte zur BASIC-Sprache werden diese Merkmale sofort für jeden, der sich des APPLESOFT bedient, erschlossen. Die farbigen graphischen Darstellungen bei niedriger und hoher Auflösung sowie die direkten Analogieangaben (bei der Spielsteuerung) gehören zu den Merkmalen, die durch APPLESOFT zum Tragen gebracht werden.

Ein weiteres Merkmal von APPLESOFT besteht darin, daß es dieses Handbuch gibt. Es ist nicht als ein Handbuch für das Selbststudium gedacht, denn dafür gibt es bei APPLE ein gesondertes Handbuch, das "BASIC" Programmierhandbuch für den APPLE II", welches Ihnen beim Erlernen des Programmierens hilft, auch wenn Sie vorher noch nie an einem Computer gesessen haben. Das vorliegende Handbuch geht von derAnnahme aus, daß Sie bereits das Programmieren in BASIC beherrschen und nur gern lernen möchten, welche zusätzlichen Möglichkeiten Ihnen durch APPLESOFT zur Verfügung stehen. Im Kapitel 1 (Einführung in den Betrieb des Computers) gehen wir kurz das durch, was Ihnen die Sprache bieten kann. Im weiteren Verlauf des Handbuches werden dann in sorgfältiger und exakter Form die einzelnen Anweisungen und deren Wirkungsweise beschrieben und erklärt. Im Bemühen, Ihnen die Enttäuschung und Verärgerung zu ersparen, die manche Handbücher verursachen können, haben wir in diesem Handbuch die Stellen hervorgehoben, wo Programmierfehler Ihnen Schwierigkeiten bereiten können. Besondere Symbole machen Sie auf diese Stellen aufmerksam.

Die Methode, die bei der Beschreibung von APPLESOFT zur Anwendung kam, stellt fast eine eigene einfache Sprache dar. Nach einigen Augenblicken der Gewöhnung werden Sie erkennen, daß sie Ihr Verständnis genau dessen, was in der Sprache erlaubt und verboten ist, beschleunigen wird. Es werden keine bohrenden Zweifel über die Ausdeutung eines Satzes bei Ihnen entstehen, wie das manchmal bei Beschreibungen in reinem Englisch der Fall sein kann.

Fortgeschrittene Programmierer werden dieses Handbuch als besonders nützlich empfinden. Angehende Programmierer werden darauf aufmerksam geamcht, daß sie bald nicht mehr zu den Anfängern gehören, und werden die besonderen Anstrengungen, die APPLE auf sich genommen hat, um ein ungewöhnlich komplettes Handbuch vorzulegen, zu schätzen wissen. Gewiß, ein dickeres Handbuch sieht gewaltiger aus, doch wenn Sie sich informieren müssen, dann werden Sie froh sein, daß wir weder Zeit noch Mühe gescheut haben, um alles Nötige aufzunehmen.

Hinweise für die Benutzung

Im vorliegenden Handbuch wird eine minimale Erfahrung beim Programmieren in der BASIC-Sprache vorausgesetzt. Kennen Sie sich nicht in BASIC aus, dann wird Ihnen das "BASIC"-Programmierhandbuch für den APPLE II eine Einführung geben. Es handelt sich dabei um eine BASIC-Version, die dem APPLESOFT II sehr ähnlich ist, aber einfacher.

Wir empfehlen Ihnen, daß Sie, wenn Sie dieses Handbuch konsultieren, das APPLESOFT-II-BASIC (im weiteren Text kurz APPLESOFT) einlegen und ablaufen lassen, so daß Sie auf Ihrem Computer alles ausprobieren können, was im Handbuch beschrieben oder vorgeschlagen wird. Wenn Ihr System mit APPLESOFT läuft, wird das Kennungszeichen des APPLESOFT ( ]) auf dem Bildschirm angezeigt. Eine Erklärung, wie man APPLESOFT in den Computer lädt, findet sich in dem Anhang A.

Es gibt zwei Ausdrücke, die Sie kennen müssen, wenn Sie das Handbuch lesen. Das Wort "syntax" bezieht sich auf die Struktur eines Computer-Befehls, die Anordnung und die richtige Form der verschiedenen Teile des Befehls. Das Wort "parse" bezieht sich auf die Art und Weise, in der der Computer versucht zu interpretieren, was sie eintasten, wobei er die verschiedenen Teile des Computer-Befehls herausgreift, um sie auszuführen. Beispielsweise gestattet Ihnen die APPLESOFT-Syntax einzutasten 12X5=4*3^2.

Wenn diese Eingabe durch APPLESOFT aufgeliedert wird, dann wird zunächst 12 als die Programmzeilennummer herausgegriffen und dann X5 als die Bezeichnung einer arithmetischen Variable interpretiert. Schließlich wird 3^2 berechnet, und das Ergebnis, nämlich 9 mit 4 multipliziert. Danach wird der Wert 36 der Variable zugeordnet, deren Bezeichnung X5 lautet.

Im Kapitel 1 findet sich eine Übersicht über viele APPLESOFT-Befehle, die für jene gedacht ist, die nur wenig Erfahrung bei der Programmierung in BASIC haben. Viele Erstkonzeptionen werden vorgestellt, wobei solche Beispiele benutzt werden, die Sie in den Computer eingeben können. In dem Anhang B wird auf das Schreiben von APPLESOFT-Programmen hingewiesen.

Die zu Beginn des Kapitel 2 eingeführte Schreibweise wird benutzt, um die Syntax von APPLESOFT in konzentrierter und eindeutiger Weise zu beschreiben. Sie wird Ihnen Zeit und Mühe sparen beim Verstehen der Art und Weise des Aufbaus des Befehls. Sie müssen diese Schreibweise nicht selbst anwenden, aber sie wird Ihnen helfen, viele Fragen, die nicht gesondert im Text erörtert werden, beantworten zu können. Beispielsweise werden eckige Klammern ([und]) verwendet, um wahlweise Teilstücke des Befehls zu kennzeichnen. Geschweifte Klammern ({ und }) schließen die Befehlsteile ein, die wiederholt werden können. So gibt [LET] C = 3 an, daß das Wort LET wahlfrei ist und ausgelassen werden kann. REM [{ character}] gibt an, daß der REMark-Befehl aus dem Wort REM besteht und wahlfrei von einem oder mehreren Zeichen gefolgt werden kann.

Die Abkürzungen und Definitionen für die Syntax im ersten Teil des Kapitel 2 werden in einer logischen Reihenfolge für diejenigen angegeben, die gern sehen möchten, wie wir unser System von Symbolen und Definitionen aufgebaut haben. Sie ziehen es vielleicht vor, diese Symbole und Definitionen außer Acht zu lassen, bis Sie im Text damit konfrontiert werden.

Die Kapitel 3 bis 1Ø geben detaillierte Erklärungen der nach Bereichen geordneten APPLESOFT-Befehle. Möchten Sie sich über einen ganz bestimmten Befehl informieren, gibt Ihnen der alphabetisch geordnete Index auf den Innenseiten des Einbandes an, wo Sie nachschlagen müssen. Zusätzliches Bezugsmaterial, das nicht in den Kapiteln behandelt wird, findet sich in den Anhängen.

An manchen Stellen werden Sie auf dieses Symbol ACHTUNG! stoßen, das jedesmal dem Abschnitt vorausgeht. Es zeigt Ihnen ein ungewöhnliches Merkmal an, auf das Sie achten sollten. Das Symbol VERBOTEN! geht einem Abschnitt voraus, der Situationen beschreibt, von denen sich APPLESOFT nicht wieder erholen kann. Sie werden Ihr Programm einbüßen und wahrscheinlich APPLESOFT wieder neu in Gang setzen müssen.

KAPITEL 1

Einführung in den Betrieb des Computers

Befehle für Sofortausführung

Geben Sie folgendes ein:

PRINT 1Ø-4

Betätigen Sie jetzt die mit RETURN beschriftete Taste. APPLESOFT II wird sofort 6 ausdrucken.

Die PRINT-Anweisung wurde in dem Augenblick, da Sie die RETURN-Taste betätigten, ausgeführt. APPLESOFT berechnete nach dem PRINT die Formel und schrieb den Wert, in diesem Falle 6, aus.

Jetzt geben Sie ein:

PRINT 1/2, 3*1Ø (Das sind zwei Aufgaben, getrennt durch das Komma).

(* heißt: Multipliziere! / heißt: Dividiere!)

Wenn Sie die RETURN-Taste betätigen, wird ausgedruckt: .5 3Ø

Wie Sie feststellen können, kann man mit APPLESOFT dividieren und multiplizieren sowie subtrahieren. Beachten Sie, wie man das Komma (,) in dem PRINT-Befehl benutzt, damit zwei Werte anstelle von nur einem Wert ausgedruckt werden. Setzt man im PRINT-Befehl das Komma ein, wird die aus 4Ø Zeichen bestehende Zeile in drei Spalten oder "Tabellenbereiche" unterteilt. Näheres zu Tabellenbereichen findet sich in Kapitel 6 unter PRINT-Befehl.

Befehle für die verzögerte Ausführung

Solche Befehle wie die PRINT-Anweisung, die Sie eben eingegeben haben, nennt man "Befehle für Sofortausführung". Eine weitere Befehlsart wird als "Befehl für verzögerte Ausführung" bezeichnet. Jeder Befehl für eine verzögerte Ausführung wird mit einer "Zeilennummer" eingeleitet. Eine Zeilennummer ist eine ganze Zahl aus dem Interval Ø - 63999.

Geben Sie folgende Zeilen ein:

```
1Ø PRINT 2+3
2Ø PRINT 2-3
```

(Achten Sie darauf, daß jede Zeile mit dem Druck auf die RETURN-Taste abgeschlossen werden muß!)

Eine Folge von Befehlen für verzögerte Ausführung nennt man ein "Programm". Vermittels APPLESOFT BASIC werden diese Befehle im Speicher des APPLE festgehalten und nicht sofort ausgeführt. Wenn Sie dann RUN eingeben, wird das APPLESOFT die gespeicherten Befehle ausführen, wobei zunächst der Befehl mit der niedrigsten Zeilennummer ausgeführt wird, dann der mit der nächsthöheren Zeilennummer usw., bis das ganze Programm erfüllt ist.

Nehmen wir an, Sie würdenjetzt RUN eintasten. (Denken Sie bitte daran, am Ende jeder Zeile RETURN zu drücken.)

RUN

Auf dem Bildschirm wird nun durch APPLESOFT angezeigt: 5
-1

Im vorangegangenen Beispiel haben wir erst Zeile 1Ø und dann Zeile 2Ø eingegeben. Es ist aber unerheblich, in welcher Reihenfolge Sie die Befehle für verzögerte Ausführung eingeben. Bei APPLESOFT wird immer die entsprechend der Zeilennummer richtige numerische Reihenfolge hergestellt.

Um das Auflisten eines augenblicklich im Speicher befindlichen kompletten Programms, in dem die Anweisungen in der richtigen Reihenfolge angeordnet sind, prüfen zu können, tasten Sie LIST ein, und APPLESOFT wird reagieren mit der Anzeige:

```
1Ø PRINT 2+3
2Ø PRINT 2-3
```

Manchmal möchte man eine ganze Zeile des Programms löschen. Das geschieht, indem man die Zeilennummer der zu löschenden Zeile eingibt und danach RETURN drückt.

Geben Sie ein:

```
1Ø
LIST
```

Daraufhin wird APPLESOFT mit der Anzeige reagieren:

```
2Ø PRINT 2-3
```

Sie haben jetzt Zeile 1Ø aus dem Programm gelöscht. Es gibt keine Möglichkeit, diese zurückzubekommen. Wenn Sie eine neue Zeile 1Ø eingeben möchten, tasten Sie einfach 1Ø ein und schreiben die neue Anweisung, die APPLESOFT erfüllen soll.

Geben Sie ein:

```
1Ø PRINT 2* 3
LIST
```

APPLESOFT wird anzeigen:

```
1Ø PRINT 2* 3
2Ø PRINT 2 - 3
```

Die Zeile 1Ø läßt sich auch einfacher ersetzen, als dadurch, daß man sie löscht und eine neue Zeile eingibt. Wenn Sie einfach die neue Zeile 1Ø eingeben (und natürlich am Ende der Zeile RETURN betätigen), wirft APPLESOFT automatisch die alte Zeile 1Ø heraus und ersetzt sie durch die neue Zeile 1Ø.

Geben Sie ein:

```
1Ø PRINT 3-3
LIST
```

APPLESOFT wird anzeigen:

```
1Ø PRINT 3-3
2Ø PRINT 2-3
```

Es empfiehlt sich nicht, in einem Programm die einzelnen Zeilen durchgängig zu numerieren, denn es könnte später einmal notwendig werden, eine weitere Zeile zwischen zwei schon vorhandenen einzugeben. Das Schrittmaß von 1Ø von Zeile zu Zeile reicht im allgemeinen aus.

Wenn Sie das komplette Programm, das sich im Speicher gerade befindet, löschen wollen, brauchen Sie lediglich NEW einzutasten. Wenn Sie ein Programm abgeschlossen haben und ein anderes beginnen möchten, denken Sie daran, zuerst NEW einzugeben, damit es nicht zu einer Mischung des alten mit dem neuen Programm kommt.

Geben Sie ein:

```
NEW
```

Auf der Anzeige erscheint das Kennungszeichen:

]

Jetzt geben Sie ein:

LIST

APPLESOFT wird Ihnen anzeigen:

]

Sie sehen, daß Ihr vorheriges Programm nicht mehr im Speicher ist.

ZAHLENSTRUKTUR

Wir müssen ein wenig weiter ausholen, um die Struktur der Zahlen, die vom APPLESOFT BASIC gedruckt werden, zu erklären. Innerhalb werden die Zahlen mit einer Genauigkeit von über 9 Stellen gespeichert. Wenn eine Zahl gedruckt wird, werden auf dem Bildschirm nur 9 Ziffern angezeigt. JedeZahl kann auch einen Exponenten haben (mit der Basis 1Ø).

Im APPLESOFT BASIC müssen die "realen genauen" Zahlen (auch Zahlen mit gleitender Kommastelle genannt) im Intervall von -1* 1Ø^ 38 bis +1* 10^ 38 liegen oder Sie laufen Gefahr, eine Fehlermeldung angezeigt zu bekommen. Beim Addieren oder Subtrahieren können Sie manchmal auf Zahlen bis 1.7 1Ø^ 38 kommen, ohne daß die Fehlermeldung kommt. Eine Zahl, deren absoluter Wert kleiner ist als ca. 3* 10^ -39, wird durch APPLESOFT zur Null umgewandelt. Außer diesen Beschränkungen gilt noch die Anforderung, daß die wahren ganzen Zahlen im Bereich von -32767 bis +32767 liegen.

Wenn eine Zahl geschrieben wird, kommen folgende Regeln zur Anwendung, um die exakte Struktur zu definieren:

1. Wenn die Zahl im negativen Bereich liegt, wird das Minuszeichen (-) gedruckt.

2. Wenn der absolute Wert einer Zahl eine ganze Zahl im Bereich von Ø bis 999999999 ist, dann wird er als ganze Zahl gedruckt.

3. Wenn der absolute Wert einer Zahl größer oder gleich Ø.Ø1 oder kleiner als

999999999.2 ist, wird die Zahl ohne Exponent mit der Festkommaschreibweise gedruckt.

4. Fällt die Zahl nicht unter die Kategorien 2 oder 3, dann kommt die wissenschaftliche Schreibeweise zur Anwendung.

Die wissenschaftliche Schreibweise wird benutzt, um reale genaue Zahlen zu drucken, und hat folgende Struktur:

SX.XXXXXXXXESTT

Dabei ist jedes X eine ganze Zahl von Ø bis 9.

Das einleitende S zeigt an, daß es sich um ein Zeichen handelt, wobei nichts vermerkt ist, wenn es sich um eine positive Zahl handelt, und ein Minuszeichen, wenn die Zahl negativ ist. Eine von Null verschiedene Ziffer steht vor dem Dezimalpunkt. Dann folgt das Komma und schließlich die anderen acht Ziffern der Mantisse. Ein E wird dann gedruckt, wo ein Exponent vorhanden ist. Diesem folgt das S als Zeichen für die Kennzeichnung als Exponent. Die danach folgenden Ziffern (TT) sind der Exponent selbst. Einleitende Nullen werden nie geschrieben, d. h. die Zahl vor dem Komma ist niemals eine Null. Auch mehrere Nullen hintereinander werden nicht geschrieben.

Muß nur eine Ziffer gedruckt werden, nachdem alle nacheinander folgenden Nullen wegfallen, wird kein Komma geschrieben. Das Exponentenzeichen für positive ist (+) und für negative (-). Zwei Ziffern des Exponenten werden immer geschrieben, d. h. Nullen werden im Exponenten nicht weggelassen.

Der Wert einerZahl, die mit Hilfe der wissenschaftlichen Schreibweise ausgedrückt wird, ist die Zahl links vom E multipliziert mit 1Ø potenziert mit der Zahl rechts vom E.

Hier einige Beispiele für verschiedene Zahlen und die Output-Struktur, die beim APPLESOFT verwendet wird, um die Zahlen zu schreiben:

Zahl	Output-Struktur
+1	1
-1	-1
6523	6523

```
-23.46Ø                                -23.46
45.72E5                                4572ØØØ
1* 1Ø^2Ø                               1E+2Ø
-12.34567896 1Ø^1Ø                     -1.2345679E+11
1ØØØØØØØØØ                             1E+Ø9
999999999                              999999999
```

Eine Zahl, die auf dem Tastenfeld eingetastet wurde, oder eine numerische Konstante, die in einem APPLESOFT-Programm benutzt wird, kann so viele Stellen haben, wie gewünscht werden, bis zu maximal 38 Stellen. Doch nur die ersten 1Ø Stellen sind meist von Bedeutung. So wird die 10. Stelle gerundet.

Wenn Sie beispielsweise eingeben:

```
PRINT 1.23456787654321,
```

dann reagiert APPLE SOFT mit

```
1.23456788
```

Beispiel für farbige graphische Darstellungen

Geben Sie ein:

```
GR
```

Auf dieses Kommando hin werden die oberen zwanzig Zeilen auf Ihrem Bildschirm dunkel. Nur vier Zeilen bleiben am unteren Rand für Text. Ihr APPLE ist jetzt auf die Betriebsart "farbige graphische Darstellungen bei niedriger Auflösung" eingestellt.

Geben Sie jetzt ein:

```
COLOR = 13
```

APPLESOFT wird nur mit der Anzeige des Bereitschaftszeichens

und des blinkenden Cursors reagieren, doch im Inneren wurde vermerkt, daß Sie ein Gelb

ausgewählt haben.

Nun geben Sie ein:

PLOT 2Ø, 2Ø

APPLESOFT wird ein kleines gelbes Quadrat in der Mitte des Bildschirms sichtbar machen. Wenn das Quadrat nicht gelb ist, dann haben Sie Ihren Apparat nicht richtig eingestellt. Stellen Sie also die Farbe und die Helligkeit richtig ein, und Sie erhalten ein leuchtendes Zitronengelb.

Geben Sie jetzt ein:

HLIN Ø, 3Ø AT 2Ø

APPLESOFT wird einen waagerechten Strich über drei Viertel des Bildschirms ziehen in einem Abstand von einem Viertel der Gesamthöhe von oben.

Geben Sie nun ein:

COLOR = 6

Damit wählen Sie eine andere Farbe. Dann tasten Sie ein:

VLIN 1Ø, 39 AT 3Ø

Mehr über farbige Darstellungen werden Sie später erfahren. Um nun wi eder auf den vollen Textbetrieb zurückzuschalten, geben Sie ein:

TEXT

Die Anordnung der Zeichen auf dem Bildschirm entspricht dann der Art und Weise, wie Ihr APPLE die Farbinformation als Text wiedergibt.

Beim PRINTen von Lösungen zu bestimmten Problemstellungen ist es oft wünschenswert, Text mit einzubeziehen, um die Bedeutung von Zahlen zu erklären.

Tasten Sie ein:

PRINT "ONE THIRD IS EQUAL TO", 1/3

APPLESOFT wird anzeigen:

```
ONE THIRD IS EQUAL TO          .333333333
```

Druckstruktur

Wie schon beschrieben, verursacht das in eine PRINT-Anweisung aufgenommene Komma eine Verschiebung in den nächsten Tabellenbereich, ehe der in der PRINT-Anweisung hinter dem Komma stehende Wert gedruckt wird. Wird statt des Komma ein Semikolon verwendet, wird der nachfolgende Wert im PRINT -Befehl unmittelbar im Anschluß an den ersten ausgedruckt. Probieren Sie es aus.

Gehen Sie folgende Beispiele durch:

```
PRINT 1,2,3
1              2              3

PRINT 1;2;3
123

PRINT -1;2;-3
-12-3
```

Im Folgenden geht es um ein Programmbeispiel, in dem der Wert vom Tastenfeld abgelesen wird und benutzt wird, um ein Ergebnis zu errechnen und auszudrucken:

```
1Ø INPUR R
2Ø PRINT 3.14159* R* R
RUN
?1Ø
314.159
```

Folgendes läuft ab. Bei der Eingabe der INPUT-Anweisung wird durch das APPLESOFT ein Fragezeichen auf dem Bildschirm angezeigt. Der Computer zeigt damit an, daß er bereit ist, eine Zahl entgegenzunehmen. Wenn Sie jetzt das tun (und im Beispiel oben wurde 1Ø eingetastet), wird der Variablen, die in der Anweisung INPUT definiert wurde, der eingetastete

Wert zugeordnet (in diesem Falle wurde die INPUT-Variable R auf 1Ø festgelegt). Das Programm läuft weiter, und Zeile 2Ø des Beispiels oben wird ausgeführt. Nachdem die Formel nach der PRINT-Anweisung ausgerechnet wurde, wird die Variable R jedesmal durch den Wert 1Ø ersetzt. So wird die Formel 3.14159* 1Ø* 1Ø oder 314.159.

Wenn Sie es noch nicht bemerkt haben sollten, das Programm berechnet den Flächeninhalt eines Kreises mit dem Radius R.

Wollten wir den Flächeninhalt verschiedener Kreise berechnen, könnten wir das Programm immer wieder durchlaufen lassen. Es geht aber noch einfacher, indem wir einfach noch die Zeile eingeben:

```
3Ø GOTO 1Ø
RUN
?1Ø
314.159
?3
28.27431
?4.7
69.3977231
?
BREAK IN 1Ø
]
```

Wenn Sie eine GOTO-Anweisung an den Schluß Ihres Programms setzen, lassen Sie es dadurch immer wieder zur Zeile 1Ø zurücklaufen, und es werden immer die jeweiligen Antworten ausgedruckt. Das läßt sich unendlich lange fortsetzen, aber wir unterbrechen die Berechnung von Kreisflächeninhalten nach dem dritten Kreis. Das Stoppen wird dadurch erreicht, daß Sie "control C" eingeben (tasten Sie das C ein, während Sie gleichzeitig die CTRL-Taste drücken) und dann die RETURN-Taste betätigen. Das löst jetzt einen "break" bei der Programmausführung aus, der uns das Stoppen ermöglicht. Wenn Sie "control C" eingeben, kann jedes beliebige Programm nach Ausführung der laufenden Anweisung angehalten werden. Nun probieren Sie selbst!

Variablennamen

Der Buchstabe R in dem eben abgelaufenen Programm wurde als "Variable" bezeichnet. Es handelt sich dabei lediglich um einen Speicherplatz in dem Computer, der durch den Namen R identifiziert wird. Ein Variablenname muß mit einem Zeichen des Alphabets beginnen, dem dann irgendein alphanumerisches Zeichen folgen kann. Bei einem alphanumerischen Zeichen handelt es sich um irgendeinen Buchstaben von A bis Z oder eine Ziffer von Ø bis 9.

Ein Variablenname kann bis zu 238 Zeichen lang sein, doch APPLESOFT benötigt nur die ersten zwei Zeichen zur Unterscheidung der Variablennamen. So beziehen sich die Namen GOOD4NOUGHT und GOLDRUSH auf die gleiche Variable.

In einem Variablennamen werden alle alphanumerischen Zeichen, die den ersten beiden Zeichen folgen, außer Acht gelassen, wenn sie nicht ein "Tabu-Wort" enthalten. Einige Worte werden in den APPLESOFT-BASIC-Kommandos für ihren speziellen Zweck "reserviert". Sie können dann diese Worte nicht als Variablennamen oder als einen Teil irgendeines Variablennamens verwenden. Beispielsweise wäre PEND als Variablenname nicht zulässig, weil END ein Tabu-Wort ist. Die im APPLESOFT-BASIC enthaltenen Tabuwörter sind im Anhang F aufgeführt und erläutert.

Variablennamen, die auf $ oder % enden, haben eine spezielle Bedeutung, die weiter unten in diesem Kapitel unter "Reale, ganze und Datenfolgenvariable" erläutert sind.

Nachstehend haben wir einige Beispiele für zulässige und unzulässige Variablennamen aufgeführt.

Zulässige	Unzulässige
TP	TO (Variablennamen dürfen keine Tabu-Wörter sein)
PSTG$	RGOTO (Variablennamen dürfen keine Tabu-Wörter sein)
COUNT	
N 1%	

Neben der Zuweisung eines Wortes zu einer Variable durch die INPUT-Anweisung gibt es auch noch die Möglichkeit, den Wert einer Variable durch eine LET- oder eine andere Zuweisungsanweisung zu setzen. Geben Sie die folgenden Beispiele ein:

```
A = 5
PRINT A, A*2
5         1Ø

LET Z = 7
PRINT Z, Z-A
7         2
```

Wie aus den Beispielen ersichtlich wird, ist das LET in einer Zuweisungsanweisung wahlfrei. BASIC "erinnert" sich der Werte, die den Variablen bei dieser Art von Anweisung zugewiesen wurden. Dieser "Erinnerungsprozeß" benötigt Platz für die Speicherung der Angaben im Speicher des APPLE.

Die Werte der Variablen werden herausgeworfen und der Speicherraum freigegeben, wenn die vier folgenden Umstände eintreten:

1. Eine neue Zeile wird in das Programm eingegeben oder eine alte Zeile wird gelöscht.

2. Das CLEAR-Kommando wird gegeben.

3. Das RUN-Kommando wird gegeben.

4. NEW wird eingetastet.

Hier kommen wir zu einer weiteren wichtigen Tatsache. Bevor Sie keinen anderen Wert zuordnen lassen, wird allen numerischen Variablen automatisch der Wert Null zugeordnet. Probieren Sie das anhand des Beispiels:

```
PRINT Q, Q+2, Q*2
Ø              2              Ø
```

Eine weitere Anweisung ist REM, das kurz für remark (bemerken) steht. Diese Anweisung läßt sich benutzen, um Kommentare oder Anmerkungen in das Programm einzufügen. Trifft BASIC auf eine REM-Anweisung, bleibt der Rest auf der Zeile unberücksichtigt. Das dient hauptsächlich als Hilfe für den Programmierer und hat keine nützliche Funktion, was die Bearbei-

tung des Programms bei der Lösung eines bestimmten Problems betrifft.

IF...THEN

Schreiben wir nun ein Programm, um zu prüfen, ob eine eingegebene Zahl den Wert Ø hat oder nicht. Keine der bisher erläuterten Anweisungen ermöglicht uns das. Was wir brauchen, ist eine Anweisung, die eine Verzweigung der Bedingung zu einer anderen Anweisung herstellt. Eine solche ist die IF...THEN-Anweisung. Geben Sie NEW ein und dann dieses Programm:

```
1Ø INPUT B
2Ø IF B = Ø THEN GOTO 5Ø
3Ø PRINT "NON-ZERO"
4Ø GOTO 1Ø
5Ø PRINT "ZERO"
6Ø GOTO 1Ø
```

Kommt für dieses Programm RUN, dann wird ein Fragezeichen auf dem Bildschirm erscheinen, welches anzeigt, daß Sie einen Wert für B eingeben möchten. Tasten Sie irgendeinen beliebigen Wert ein. Der Computer wird nun zur IF-Anweisung vorgehen. Zwischen dem IF- und dem THEN-Teil der Anweisung liegt eine "Behauptung". Eine Behauptung besteht aus zwei durch eines der nachstehend aufgeführten Symbole getrennten Ausdrücken:

=	gleich
>	größer als
<	kleiner als
<> oder ><	nicht gleich
(Symbol)	(Bedeutung)
<=	kleiner als oder gleich
>=	größer als oder gleich

Die IF-Anweisung ist entweder wahr oder falsch, je nachdem, ob die Behauptung wahr oder falsch ist. In dem vorliegenden Programm ist die Behauptung B=Ø zum Beispiel wahr, wenn Ø für B eingegeben wird. Deshalb ist die IF-Anweisung wahr, und der Programmablauf wird dem

THEN-Teil der Anweisung fortgesetzt, der lautet GOTO 5Ø. Im Anschluß an diesen Befehl springt der Computer auf die Zeile 5Ø, und ZERO wird geschrieben. Der GOTO-Befehl in Zeile 6Ø schickt den Computer zurück nach Zeile 1Ø.

Nehmen wir an, für B wird 1 eingegeben. Da die Behauptung B=Ø jetzt falsch ist, ist auch die IF-Anweisung falsch, und das Programm läuft weiter, ohne den THEN-Teil der Anweisung oder irgendeine andere Anweisung auf der entsprechenden Zeile zu beachten. Deshalb wird NON-ZERO geschrieben. Das GOTO auf Zeile 4Ø wird jetzt den Computer zur Zeile 1Ø zurückschicken.

Probieren Sie jetzt das folgende Programm für den Vergleich von zwei Zahlen aus. (Denken Sie daran. NEW einzugeben, damit das letzte Programm gelöscht wird.)

```
1Ø INPUT A,B
2Ø IF A <= B THEN GOTO 5Ø
3Ø PRINT "A IS LARGER"
4Ø GOTO 1Ø
5Ø IF A<B THEN GOTO 8Ø
6Ø PRINT "THEY ARE THE SAME"
7Ø GOTO 1Ø
8Ø PRINT "B IS LARGER"
9Ø GOTO 1Ø
```

Kommt jetzt das RUN, wird auf Zeile 1Ø ein Fragezeichen angezeigt. Sie sollen nun zwei durch ein Komma getrennte Zahlen eintasten. Wenn A > B, dann ist A <= B falsch, und das THEN GOTO 5Ø wird außer Acht gelassen. Der Programmablauf springt auf die Anweisung über, die der nächsten Zeilennummer folgt, wobei A IS LARGER gedruckt wird und das Kommando auf Zeile 4Ø den Computer zur Zeile 1Ø, also zum Start, zurückschickt.

Wenn A den gleichen Wert wie B hat, ist A <= B auf Zeile 2Ø wahr, so daß THEN GOTO 5Ø ausgeführt wird und der Computer auf Zeile 5Ø geschickt wird. Da A und B den gleichen Wert haben, ist die Aussage A < B falsch, woraufhin das THEN GOTO 8Ø außer Acht gelassen wird und der Computer zur folgenden Zeile übergeht, wo ihm gesagt wird, daß er THEY ARE THE SAME drucken soll. Die Zeile 70 schickt ihn schließlich zum Start zurück.

Wenn A < B, dann ist die Aussage A <= B auf Zeile 2Ø wahr, und der nächste Schritt, das THEN GOTO 5Ø, wird ausgeführt. Die Aussage A < B auf Zeile 5Ø trifft zu, so das THEN GOTO 8Ø ausgeführt wird. Schließlich wird B IS LARGER geschrieben, und der Computer kehrt zur Anfangszeile zurück.

Lassen Sie die letzten beiden Programme mehrmals durchlaufen. Versuchen Sie dann, Ihr eigenes Programm mit dem IF...THEN-Befehl zu schreiben. Das Probieren mit eigenen Programmen ist die schnellste und einfachste Methode, die Art und Weise, wie APPLESOFT-BASIC funktioniert, zu begreifen. Denken Sie daran, daß Sie nur "control C" eintasten und die RETURN-Taste drücken müssen, um die Programme anzuhalten.

Ein weiteres Farbbeispiel

Versuchen wir jetzt einmal ein Programm für graphische Darstellungen. Beachten Sie den Gebrauch der REM-Anweisungen zur größeren Klarheit. Der Doppelpunkt (:) wird verwendet, um Mehrfachinstruktionen auf einer numerierten Programmzeile zu trennen. Nachdem sie das unten stehende Programm eingetastet haben, lassen Sie es auflisten. Versichern Sie sich, daß Sie es richtig eingegeben haben. Danach lassen Sie es ablaufen.

```
1ØØ/GR : REM  SET  COLOR  GRAPHICS  MODE
11Ø HOME : REM  CLEAR  TEXT  AREA
12Ø X = Ø : Y = 5 : REM  SET  STARTING  POSITION
13Ø XV = 2 : REM  SET  X  VELOCITY
14Ø YV = 1 : REM  SET  Y  VELOCITY
15Ø REM CALCULATE NEW  POSITION
16Ø NX = X + XV : NY = Y + YV
17Ø REM IF BALL EXCEEDS SCREEN EDGE, THEN BOUNCE
18Ø IF NX > 39 THEN NX = 39 : XV = - XV
19Ø IF NX < Ø THEN NX = Ø : XV = -XV
2ØØ IF NY > 39 THEN NY = 39 : XV = -YV
21Ø IF NY < Ø THEN NY = Ø : YV = -YV
22Ø REM PLOT NEW POSITION IN YELLOW
23Ø COLOR = 13 : PLOT NX, NY
24Ø REM ERASE OLD POSITION
```

```
25Ø COLOR = Ø: PLOT X,Y
26Ø REM SAVE CURRENT POSITION
27Ø X = NX : Y = NY
28Ø REM STOP AFTER 250 MOVES
29Ø I = I + 1 : IF I < 25Ø THEN GOTO 16Ø
3ØØ PRINT "TO RETURN TO YOUR PROGRAMM, TYPE 'TEXT'"
```

Der GR-Befehl sagt dem APPLE, daß er auf Farbbetrieb für graphische Darstellungen umschalten soll. Gleichzeitig wird auch der 4Ø x 4Ø große Raum für das Setzen von Punkten freigemacht. Der Raum für Text-Output wird auf vier Zeilen zu je 4Ø Zeichen an der unteren Begrenzung des Bildschirms beschränkt. GR setzt automatisch color = schwarz.

HOME wird benutzt, um den Textbereich freizumachen und den "cursor" in die linke obere Ecke des gegenwärtigen Textfeldes zu bringen. Im Farb-GR-Mode würde das dem Anfang der Zeile 2Ø entsprechen, denn Zeile Ø bis 19 bilden jetzt das Feld für die farbigen graphischen Darstellungen.

Die COLOR=-Befehle in den Zeilen 23Ø und 25Ø setzen die nächste zu druckende Farbe auf den Wert des Ausdrucks, der dem COLOR= folgt.

Auf die Anweisung PLOT NX,NY in Zeile 23Ø hin wird ein kleines Quadrat an der neuen Position, die durch die Ausdrücke NX und NY gekennzeichnet sind, sichtbar. Es erscheint in der Farbe, die von dem zuletzt gegebenen COLOR=-Befehl bestimmt wird. Denken Sie daran, daß NX und NY jeweils Zahlen aus dem Bereich Ø bis 19 sein müssen, denn anderenfalls wird das Quadrat außerhalb des Bildschirms liegen und eine Fehlermeldung würde erscheinen.

In gleicher Weise wird durch das PLOT X,Y in Zeile 25Ø ein Quadrat an der durch die Ausdrücke X und Y identifizierten Stelle gedruckt. X und Y sind aber nur die "alten" Koordinaten NX und NY, die gespeichert wurden, nachdem das vorherige gelbe Quadrat eingetragen wurde. Deshalb wird durch PLOT X,Y das "alte" gelbe Quadrat überdeckt von einem Quadrat, dessen Farbe durch COLOR=Ø definiert wurde. Diese Farbe ist schwarz, also dem Untergrund, auf dem das Quadrat aufgetragen wurde, gleich. Es scheint demnach, als sei das "alte" gelbe Quadrat gelöscht.

Beachten Sie: Um von graphischen Farbdarstellungen zum Textmode umzuschalten, tasten Sie

TEXT

ein und betätigen dann die RETURN-Taste. Durch das Eingeben von TEXT, wie es vorgeschrieben ist, schalten Sie sich aus dem GR-Mode aus. Lassen Sie sich nicht von den eigenartigen Symbolen auf dem Bildschirm verwirren. Sie kommen durch das Umschalten von graphischen Darstellungen auf Textzeichen zustande. Falls Sie Zeile 29Ø nicht verstehen sollten, werden Sie nicht ungeduldig. Sie wird auf den folgenden Seiten erklärt. Wie Sie gesehen haben, kann der APPLE mehr, als nur mit Zahlen spielen. Wir kommen noch einmal auf die farbigen graphischen Darstellungen zurück, nachdem Sie noch mehr über APPLESOFT-BASIC erfahren haben.

FOR ... NEXT

Ein Vorteil der Computer besteht in ihrer Fähigkeit, sich wiederholende Aufgaben zu lösen. Stellen Sie sich vor, wir wollen eine Tabelle von Quadratwurzeln für die ganzen Zahlen von 1 bis 1Ø aufstellen. Die APPLESOFT-BASIC-Funktion heißt SQR, deren Form SQR (X) lautet, wobei X die Zahl ist, deren Quadratwurzel Sie berechnen wollen. Wir könnten das Programm folgendermaßen schreiben.

```
 1Ø PRINT 1, SQR (1)
 2Ø PRINT 2, SQR (2)
 3Ø PRINT 3, SQR (3)
 4Ø PRINT 4, SQR (4)
 5Ø PRINT 5, SQR (5)
 6Ø PRINT 6, SQR (6)
 7Ø PRINT 7, SQR (7)
 8Ø PRINT 8, SQR (8)
 9Ø PRINT 9, SQR (9)
1ØØ PRINT 1Ø, SQR (10)
```

Dieses Programm reicht aus, doch es ist fürchterlich ineffektiv. Wir können das Programm erheblich verbessern, indem wir die IF-Anweisung, die wir gerade eingeführt haben, zur Anwendung bringen:

```
10 N = 1
20 PRINT N, SQR (N)
30 N = N + 1
40 IF N <= 10 GOTO 20
```

Wenn Sie RUN geben, dann wird der Output des Programms genau wie der des Programms mit 10 Anweisungen, aussehen. Untersuchen wir, wie es arbeitet. In Zeile 10 wird eine LET-Anweisung gegeben, die die Variable N auf dem Wert 1 festsetzt. In Zeile 20 wird der Computer angewiesen, N und die Quadratwurzel von N zu drucken, wobei jedesmal der laufende Wert von N benutzt wird. So sieht Zeile 20 dann so aus:

```
20 PRINT 1, SQR (1)
```

Das Ergebnis dieser Berechnung wi rd ausgedruckt. In Zeile 30 wird eine auf den ersten Blick scheinbar ungewöhnliche LET-Anweisung gegeben. Mathematisch ist N = N + 1 unsinnig. Doch das wesentliche, woran Sie denken müssen, ist, daß in einer LET-Anweisung das Symbol "=" kein Gleichheitszeichen ist, sondern in diesem Falle bedeutet es "soll ersetzt werden durch". Die Anweisung nimmt einfach den laufenden Wert von N auf und addiert 1. So wird N, nachdem die Zeile 30 das erste Mal durch ist, 2.

In Zeile 40 ist die Behauptung N < = 10 wahr, da ja N nunmehr 2 ist, und der THEN-Teil der Anweisung schickt den Computer zur Zeile 20 zurück, wobei N jetzt dem Wert 2 entspricht. Insgesamt geschieht nichts Anderes, als daß die Zeilen 20 bis 40 wiederholt werden, wobei jedesmal 1 zum Wert N addiert wird. Wenn N schließlich 10 in Zeile 20 ist, wird es durch die nächste Zeile zu 11. Damit wird die Behauptung in Zeile 40 falsch, und der THEN-Teil der Anweisung wird unberücksichtigt gelassen. Da keine weiteren Anweisungen vorliegen, hält das Programm an.

Dieses Verfahren ist als "Durchlaufen" oder "Interation" bekannt. Da es beim Programmieren äußerst häufig zur Anwendung kommt, gibt es dafür im BASIC spezielle Anweisungen. Wir zeigen sie im folgenden Programm:

```
10 FOR N = 1 TO 10
20 PRINT N, SQR (N)
30 NEXT N
```

Der Output des oben angeführten Programms sieht ganz genau so aus, wie der der beiden vorangegangenen Programme. In Zeile 1Ø wird N gleichgesetzt mit 1. Durch Zeile 2Ø wird der Wert für N und die Quadratwurzel von N geschrieben. Eine neue Anweisung finden wir in Zeile 3Ø. Die Anweisung NEXT N hat zur Folge, daß 1 zu N addiert wird und daß, wenn N <= 1Ø, die Programmausführung wieder mit der Anweisung, die dem FOR folgt, beginnt. In diesem Falle gibt es nichts Besonderes zu N zu sagen. Es kann jede Variable genommen werden, solange sie nicht von dem Variablennamen in der FOR oder NEXT-Anweisung abweicht. Beispielsweise könnte man überall in dem o. g. Programm N durch Z1 ersetzen, und es würde in gleicher Weise funktionieren.

Stellen Sie sich vor, wir wollten eine Tabelle der Quadratwurzeln aller geraden ganzen Zahlen von 1Ø bis 2Ø aufstellen. Mit dem folgenden Programm läßt sich diese Aufgabe erfüllen:

```
1Ø N = 1Ø
2Ø PRINT N, SQR (N)
3Ø N = N+2
4Ø IF N <= 2Ø THEN GOTO 2Ø
```

Beachten Sie die gleiche Struktur dieses Programms und des Programms für das Schreiben der Quadratwurzeln der Zahlen 1 bis 1Ø. Das Programm läßt sich auch mit Hilfe der eben eingeführten FOR-Schleife schreiben:

```
1Ø FOR N = 1Ø TO 2Ø STEP 2
2Ø PRINT N, SQR (N)
3Ø NEXT N
```

Beachten Sie, daß der wichtigste Unterschied zwischen diesem Programm und dem vorherigen, worin die FOR-Schleife zur Anwendung kam, darin besteht, daß im zweiten STEP 2 hinzugefügt wurde. Das zeigt dem APPLESOFT an, 2 zu jedem N zu addieren, und nicht 1, wie das im vorigen Programm der Fall war. Wenn kein STEP für die FOR-Anweisung gegeben wird, wird durch APPLESOFT vorausgesetzt, daß jedesmal 1 addiert werden soll. Dem STEP kann jeder beliebige Ausdruck folgen. Nehmen wir jetzt an, wir wollten von 1Ø bis 1 rückwärts zählen. Ein dafür geeignetes Programm wäre :

```
1Ø I = 1Ø
2Ø PRINT I
3Ø I = I - 1
4Ø IF I>= 1 THEN GOTO 2Ø
```

Beachten Sie, daß wir jetzt prüfen, um dafür zu sorgen, daß I größer als oder gleich dem Schlußwert ist. Der Grund dafür ist, daß wir jetzt mit einer negativen Zahl zählen. In den vorangeganenen Beispielen war es das Gegenteil, so daß wir eine Variable prüften, ob sie kleiner als oder gleich dem Schlußwort war.

Die STEP-Anweisung, die schon vorher gezeigt wurde, kann auch bei negativen Zahlen zum gleichen Zweck benutzt werden. Das geschieht durch die Verwendung der gleichen Druckstruktur wie in dem anderen Programm, nämlich:

```
1Ø FOR I = 1Ø TO 1 STEP -1
2Ø PRINT I
3Ø NEXT I
```

Die FOR-Schleifen lassen sich auch aneinanderfügen. Ein Beispiel dafür:

```
1Ø FOR I = 1 TO 5
2Ø FOR J = 1 TO 3
3Ø PRINT I, J
4Ø NEXT J
5Ø NEXT I
```

Beachten Sie, daß NEXT J dem NEXT I vorausgeht, denn die J-Schleife liegt innerhalb der I-Schleife. Das folgende Programm ist nicht korrekt. Geben Sie RUN, und Sie werden sehen, was geschieht.

```
1Ø FOR I = 1 TO 5
2Ø FOR J = 1 TO 3
3Ø PRINT I, J
4Ø NEXT I
5Ø NEXT J
```

Es funktioniert nicht, weil alle Kenntnisse über die J-Schleife verlorengegangen sind, wenn

NEXT I eingegeben wird.

Bereiche

Es macht sich oft sehr gut, irgendein Element aus einer Zahlentabelle auswählen zu können. Beim APPLESOFT wird das durch die Verwendung von Bereichen möglich. Ein Bereich ist eine Zahlentabelle. Der Name dieser Tabelle, der Bereichsname genannt wird, ist jeder beliebige zulässige Variablenname, A beispielsweise. Der Bereichsname A unterscheidet sich vom einfachen Variablennamen A und ist ein selbständiger Name. Sie können also beide in einem einzigen Programm verwenden.

Um ein Element der Tabelle auswählen zu lassen, versehen wir A mit einem Index. Wenn wir also das I-te auswählen lassen wollen, klammern wir I ein (I) und schreiben den Ausdruck hinter das A. So bezeichnet A (I) das I-te Element im Bereich A.

Beachten Sie!
In diesem Kapitel des Handbuchs werden nur die eindimensionalen Bereiche behandelt. Weitere Informationen und Erläuterungen der APPLESOFT-Befehle für Bereiche finden sich in Kapitel 5 "Bereiche und Datenfolgen".

A (I) ist nur ein Element des Bereichs A. Im APPLESOFT muß vermerkt sein, wieviel Platz für den gesamten Bereich zuzuordnen ist, d. h. welches die maximale Größe des Bereichs sein wird. Das geschieht mit Hilfe der DIM-Anweisung,
wobei die Form DIM A (15) genommen wird. In diesem Falle haben wir Platz für den Bereichsindex reserviert, der sich von Ø bis 15 erstreckt. Bereichsindizes beginnen immer bei Ø, weshalb im o. g. Beispiel für den Bereich 16 Zahlen zulässig sind.

Wenn A (I) in einem Programm verwendet wurde, bevor es geDIMed wurde, wird durch APPELSOFT Raum für 11 Elemente freigehalten (also für Indizes von Ø bis 1Ø).
Als Beispiel dafür, wie die Bereiche angewendet werden, probieren Sie das folgende Programm durch, in dem eine Liste von Ihnen eingegebenen Zahlen sortiert wird.

```
9Ø DIM A (8): REM DIMENSION ARRAY WITH MAX. 9 ELEMENTS
1ØØ REM ASK FOR 8 NUMBERS
11Ø FOR I = 1 TO 8
```

```
120 PRINT "TYPE A NUMBER:  ";
130 INPUT A (I)
140 NEXT I
150 REM PASS THROUGH 8 NUMBERS, TESTING BY PAIRS
160 F = 0 : REM RESET THE ORDER INDICATOR
170 FOR I = 1 TO 7
180 IF A (I) < = A(I+1) THEN GOTO 140
190 REM INTERCHANGE A (I) AND A (I+1)
200 T = A (I)
210 A (I) = A (I+1)
220 A (I+1) = T
230 F = 1 : REM ORDER WAS NOT PERFECT
240 NEXT I
250 REM F = 0 MEANS ORDER IS PERFECT
260 IF F = 1 THEN GOTO 160 : REM TRY AGAIN
270 PRINT : REM SKIP A LINE
280 REM PRINT ORDERED NUMBERS
290 FOR I = 1 TO 8
300 PRINT A (I)
310 NEXT I
```

Bei Ausführung von Zeile 90 wird durch APPLESOFT Raum für 9 Zahlenwerte, A (0) bis A (8), freigehalten. In den Zeilen 110 bis 140 kommt die unsortierte Liste vom Benutzer. Das Sortieren selbst geschieht in den Zeilen 170 bis 240, indem der Computer durch die Zahlenliste geht und immer wechselt bei jeder zweiten Zahl, die nicht in der Reihenfolge ist. F bedeutet "Anzeige der richtigen Reihenfolge". F = 1 zeigt an, daß umgeschaltet wurde. Wenn welche gemacht wurden, wird dem Computer in Zeile 260 angewiesen, noch einmal zurückzugehen, und weiter zu prüfen.

Wenn durch alle acht Zahlen durchgegangen wurde, ohne daß es zu Austauschen kam (was bedeutet, daß die Zahlen alle in der Reihenfolge standen), werden die Zeilen 290 bis 310 das Ausschreiben der sortierten Liste veranlassen. Denken Sie daran, daß ein Index jeder beliebige Ausdruck sein kann.

GOSUB...RETURN

Ein weiteres sehr nützliches Anweisungspaar ist GOSUB...RETURN. Wenn in Ihrem Programm die gleichen Operationen an mehreren verschiedenen Stellen durchgeführt werden, können Sie die GOSUB- und die RETURN-Anweisung benutzen, um zu vermeiden, alle Anweisungen für die gleichen Operationen an jeder Stelle des Programms noch einmal zu schreiben.

Wenn bei Ausführung des Programms ein GOSUB auftritt, dann verzweigt APPLESOFT zu der Zeile, hinter deren Nummer das GOSUB steht. Doch APPLESOFT "vergißt" dabei nicht, wie weit es das Programm bereits ausgeführt hat, bevor die Umschaltung erfolgte. Kommt jetzt die RETURN-Anweisung, dann kehrt APPLESOFT zu der ersten Anweisung nach dem letzten GOSUB, das ausgeführt wurde, zurück. Betrachten Sie das folgende Programm:

```
20 PRINT "WHAT IS THE FIRST NUMBER";
30 GOSUB 100
40 T = N : REM SAVE INPUT
50 PRINT "WHAT IS THE SECOND NUMBER";
60 GOSUB 100
70 PRINT "THE SUM OF THE TWO NUMBERS IS" ; T + N
80 STOP : REM END OF THE MAIN PROGRAM
100 INPUT N : REM BEGIN   INPUT SUBROUTINE
110 IF N = INT (N) THEN GOTO140
120 PRINT "SORY, NUMBER MUST BE AN INTEGER. TRY AGAIN."
130 GOTO 100
140 RETURN : REM END OF SUBROUTINE
```

In diesem Programm wird nach zwei ganzen Zahlen gefragt, deren Summe dann ausgedruckt wird. Das Unterprogramm in diesem Programm reicht von Zeile 100 bis 140. Im Unterprogramm wird nach einer Zahl gefragt. Wenn es sich bei der dann eingegebenen Zahl um eine ganze Zahl handelt, wird nochmals eine Zahl gefordert. Es wird solange nach einer Zahl gefragt, bis ein ganzzahliger Wert eingegeben wird.

Das Hauptprogramm druckt WHAT IS THE FIRST NUMBER aus und fordert dann über das Unterprogramm den Wert der Zahl N an. Wenn aus dem Unterprogramm das RETURN kommt (in Zei-

le 4Ø), wird die eingetastete Zahl in der Variablen T gespeichert. Das geschieht deshalb, damit der Wert der ersten Zahl nicht verloren geht, wenn das Unterprogramm das zweitemal angefordert wird. Dann wird WHAT IS THE SECOND NUMBER ausgedruckt und das Unterprogramm erneut gerufen, jetzt, um die zweite Zahl zu bekommen.

Wenn das Unterprogramm das zweite Mal RETURNt(in Zeile 7Ø), wird THE SUM OF THE TWO NUMBERS IS ausgedruckt. Im Anschluß an den Text steht der Wert der Summe der beiden Zahlen. T ist der Wert der ersten eingetasteten Zahl und N ist der Wert der zweiten.

Die nächste ist die STOP-Anweisung in diesem Programm. Dadurch wird die Programmausführung bei Zeile 8Ø gestoppt. Erschiene die STOP-Anweisung nicht, dann würde die Programmausführung "in das Unterprogramm" auf Zeile 1ØØ fallen". Das möchten wir nicht, denn wir würden wieder nach einer weiteren Zahl gefragt werden. Würden wir jetzt eine Zahl eingeben, dann würde das Unterprogramm das RETURN auszuführen versuchen, doch da kein GOSUB kam, das das Unterprogramm aufforderte, würde es zu einem Fehler kommen. Jedes GOSUB in einem Programm sollte ein dazu passendes RETURN haben, das später durchgeführt werden kann, und jedes RETURN sollte nur dann in einem Programm stehen, wenn vorher auch das GOSUB vorhanden ist, das das Unterprogramm abruft.

STOP oder END können zur Trennung eines Programms vom Unterprogramm genommen werden. STOP wird melden, auf welcher Zeile das STOP kam, END beendet das Programm ohne Meldung. Beide Befehle übergeben die Steuerung an den Benutzer und lassen das Kennungszeichen] und den blinkenden Cursor auf dem Bildschirm erscheinen.

READ ... DATA...RESTORE

Nehmen wir an, Sie möchten gern in Ihrem Programm Zahlen verwenden, die nicht jedesmal, wenn das Programm läuft, verändert werden, aber dennoch, falls nötig, leicht veränderbar sein sollen.

Im BASIC gibt es dafür besonders Anweisungen, die man READ- und DATA-Anweisungen nennt.

Schauen Sie sich das folgende Programm an:

```
1Ø PRINT "GUESS A NUMBER";
2Ø INPUT G
3Ø READ D
4Ø IF D = -999999 THEN GOTO 9Ø
5Ø IF D<> G THEN GOTO 3Ø
6Ø PRINT "YOU ARE CORRECT"
7Ø END
9Ø PRINT "BAD GUESS, TRYAGAIN!"
95 RESTORE
1ØØ GOTO 1Ø
11Ø DATA 1, 393, - 39, 28, 391, -8, Ø, 3.14, 9Ø
12Ø DATA 89, 5, 1Ø, 15, -34, -999999
```

Das geschieht nun, wenn Sie das Programm ablaufen lassen: Trifft der Computer auf die READ-Anweisung, kommt es zu den gleichen Wirkungen wie bei einem INPUT-Befehl, doch statt eine Zahl über das Tastenfeld eingegeben zu bekommen, entnimmt der Computer die Zahl der DATA-Anweisungen.

Wenn das erstemal eine Zahl für ein READ benötigt wird, wird die ersteZahl aus der DATA-Anweisung wiedergegeben, beim zweitenmal die zweite Zahl aus der ersten DATA-Anweisung. Ist der Inhalt der ersten DATA-Anweisung auf diese Art und Weise vollständig abgefragt, wird die zweite DATA-Anweisung benutzt. Die DATA werden immer in Reihenfolge gelesen, und Ihr Programm kann eine beliebige Zahl von DATA-Anweisungen enthalten.

Der Sinn und Zweck dieses Programms ist ein kleines Spiel, in dem Sie eine der in den DATA-Anweisungen enthaltenen Zahlen erraten sollen. Jedesmal, wenn Sie eine geratene Zahl eintasten, geht der Computer durch alle in den DATA-Anweisungen enthaltenen Zahlen, bis er eine findet, die der geratenen entspricht. Wenn bei READ -999999 erscheint, sind alle in den DATA enthaltenen Zahlen abgefragt worden, und eine weitere angenommene Zahl muß eingegeben werden.

Bevor der Computer wieder zur Zeile 1Ø zurückgeht, damit wir neu raten können, müssen wir dafür sorgen, daß beim READ von Beginn der DATA-Anweisung an gelesen wird. Dazu benutzen wir den Befehl RESTORE. Nach dem RESTORE wird die nächste Angabe, bei der

READ ausgeführt wird, die erste in der ersten DATA-Anweisung sein. Die DATA-Anweisung kann sich an beliebiger Stelle im Programm befinden. Nur durch READ werden die DATA-Anweisungen bzw. deren Inhalt benutzt, ansonsten werden siebeim Ablauf eines Programms außer Acht gelassen.

Reale, ganze und Datenfolgenvariabele

Es werden drei verschiedene Arten von Variablen im APPLESOFT-BASIC verwendet. Bis jetzt haben wir nur eine Art eingesetzt, nämlich die realen Präzisionsvariablen. Zahlen bei dieser Operationsweise werden mit bis zu neunkommastelliger Genauigkeit angezeigt und können sich bis etwa 10^{38} erstrecken. APPLESOFT wandelt die von Ihnen eingegebenen Zahlen für den internen Gebrauch vom Dezimal- in das Binärsystem um und wieder zurück in das Dezimalsystem , wenn Sie das durch PRINT anordnen. Wegen der Rundungsfehler und anderer nicht vorhersehbarer Ungenauigkeiten werden durch die internen mathematischen Programme, wie Wurzelziehen, Dividieren und Potenzieren, nicht immer die exakten Zahlen angegeben, die Sie erwartet haben.

Die Anzahl der Stellen hinter dem Komma kann vor dem Ausdrucken durch die Rundung des Wertes gesetzt werden. Die allgemeine Formel für einen solchen Schritt hat folgendes Aussehen:

```
X = INT (X*1Ø^D+.5) INT (1Ø^D+.5)
```

In diesem Falle ist D die Zahl der Stellen hinter dem Komma. Schneller ginge das Setzen der Anzahl der Stellen hinter dem Komma, indem man die Stellen hinter dem Komma P = 10 ^ D sein läßt und diese Formel verwendet:

```
X = INT (X*P+.5)/P
```

Hier bedeutet P = 1Ø eine Stelle hinter dem Komma, P = 1ØØ zwei, P = 1ØØØ drei usw. Die o. g. Verfahrensweise gilt für X >= 1 und X < 999999999. Ein Maschinenprogramm zur Begrenzung der Anzahl der Stellen hinter dem Komma findet sich im nächsten Abschnitt dieses Kapitels.

Die nachstehend aufgeführte Tabelle faßt die drei Variablentypen, die bei der Programmierung mit APPELSOFT-BASIC zur Anwendung kommen, zusammen.

Beschreibung	Symbol- dem Variablen-namen hinzuzufügen	Beispiel
Datenfolgen (Ø bis 255 Zeichen	$	ALPHA$ A$
Ganze Zahlen (müssen im Bereich -32767 bis +32767 liegen)	%	B% C1%
Reale Präzisionszahlen (Exponent -38 bis +38, neun Stellen hinter dem Komma)	keines	C BOY

Einer ganzen oder Datenfolgenvariable muß immer ein % bzw. $ folgen. Beispielsweise sind X, X% und X$ unterschiedliche Variable. Ganze Variable sind in FOR oder DEF-Anweisungen unzulässig. Der größte Vorteil der ganzen Variablen besteht darin, daß sie in Bereichsoperationen, wo immer das möglich ist, eingesetzt werden können, um Speicherraum zu sparen.

Alle arithmetischen Operationen werden mit realen Präzisionsvariablen ausgeführt. Die ganzen Zahlen oder die Werte der ganzen Variablen werden in reale Präzisionszahlen umgewandelt, bevor sie in einer Berechnung verwendet werden. Bei den Sinus-, Cosinus-, Arcustangens-, Tangens-, Quadratwurzel-, Logarithmus, den Potenz-, und den Zufallsberechnungsoperationen werden die Argumente auch in reale Präzisionszahlen umgewandelt. Ergebnisse werden auch in dieser Weise ausgegeben.

Wenn eine Zahl zu einer ganzen Zahl umgewandelt wird, wird sie verfälscht (abgerundet), z. B.:

```
1%= .999        A% = -.Ø1
PRINT 1%        PRINT A%
Ø               -1
```

Wenn Sie einer realen Zahl eine ganze Variable zuordnen und dann durch PRINT den Wert der ganzen Variable schreiben lassen, geschieht das, als ob die INT-Operation angewandt

worden wäre. Es wird nicht automatisch von Datenfolge in Zahl und umgekehrt verwandelt. Ordnen Sie einer Datenfolgevariable eine Zahl zu, dann erhalten Sie eine Fehlermeldung. Es gibt jedoch spezielle Arbeitsweisen, um von einer Art zur anderen zu kommen.

Datenfolgen

Eine Folge von Zeichen wird "Literal" bezeichnet. Eine "Datenfolge" (string) ist ein in Anführungsstriche eingeschlossenes Literal. Das sind alles Datenfolgen:

```
"BILL"
"APPLE"
"THIS IS A TEST"
```

Den Datenfolgevariablen lassen sich wie den numerischen Variablen besondere Werte zuordnen. Die Datenfolgenvariablen unterscheiden sich von den numerischen Variablen durch das $ im Anschluß an den Variablennamen. Probieren Sie zum Beispiel:

```
A$ = "GOOD MORNING"
PRINT A$
GOOD MORNING
```

In diesem Beispiel haben wir den Datenfolgenwert "GOOD MORNING" für die Datenfolgenvariable A$ gesetzt. Jetzt, da wir einen Datenfolgenwert für A$ gesetzt haben, können wir leicht feststellen, wie lang dieser Wert ist, d. h. wieviel Zeichen er enthält. Das geschieht folgendermaßen:

```
PRINT LEN (A$), LEN ("YES")
12              3
```

Die LEN-Operation gibt eine ganze Zahl wieder, die der Anzahl der Zeichen in einer Datenfolge entspricht, d. h. ihre LENgth (Länge). Die Anzahl der Zeichen in einem Datenfolgenausdruck kann von Ø bis 255 reichen. Eine Datenfolge, die kein Zeichen enthält, wird als "Null-Datenfolge" bezeichnet. Ehe einer Datenfolgevariable im Programm ein Wert zugeordnet wird, wird sie anfänglich der Nullgruppe zugeordnet. Wenn man vermittels des PRINT am Schluß eine Nulldatenfolge schreiben läßt, werden keine Zeichen gedruckt, und der Cursor rückt nicht zur nächsten Spalte vor. Versuchen Sie jetzt das:

```
PRINT LEN (Q,$); Q$; 3
Ø3
```

Sie können aber auch auf eine andere Art und Weise eine Nulldatenfolge aufbauen, indem Sie

Q$ = '''' oder die äquivalente Anweisung LET Q$ = ''''

verwenden.

Das Setzen einer Datenfolgenvariable für die Nulldatenfolge kann benutzt werden, um den Datenfolgenraum, der von einer Nicht-Datenfolgenvariable belegt ist, freizumachen. Sie können dabei aber in Schwierigkeiten kommen, wenn Sie der Datenfolgenvariable eine Nulldatenfolge zuordnen, wie das in Kapitel 7 unter dem Thema IF-Anweisung erörtert wird.

Es kommt häufig vor, daß man gern einen Teil der Datenfolge herausgreifen möchte, um damit zu arbeiten. Jetzt, da wir die A$ auf "GOOD MORNING" festgesetzt haben, könnten wir nur die ersten vier Zeichen ausdrucken lassen. Dazu würden wir so verfahren:

```
PRINT LEFT$(A$,4)
GOOD
```

Das Kommando LEFT$ (AS,N) gilt für eine Datenfolgenoperation, bei der eine Unterdatenfolge, bestehend aus N, von der linken Seite aus gezählten Zeichen des Datenfolgenarguments, in diesem Falle A$, wiedergegeben wird. Hier ein weiteres Beispiel:

```
FOR N = 1 TO LEN(A$) : PRINT LEFT$(A$,N) : NEXT N
G
GO
GOO
GOOD
GOOD
GOOD M
GOOD MO
GOOD MOR
GOOD MORN
GOOD MORNI
GOOD MORNIN
```

GOOD MORNING

Da A$ über 12 Zeichen verfügt, wird diese Schleife ausgeführt mit N = 1, 2, 3, ..., 11, 12. Bei der ersten Schleife wird nur das erste Zeichen gedruckt, beim zweiten Mal die ersten beiden usw.

Es gibt eine weitere Datenfolgenoperation, nämlich RIGHT$. Das RIGHT (A$,N) gibt N Zeichen, von rechts gezählt, des Datenfolgenausdrucks wieder. Versuchen Sie jetzt, LEFT$ durch RIGHT$ im o. g. Beispiel zu ersetzen, und sehen Sie, was geschieht. Es gibt auch eine Datenfolgenoperation, mit der es uns möglich wird, Zeichen aus der Mitte der Datenfolge herauszulösen. Versuchen Sie jetzt folgendes:

```
FOR N = 1 TO LEN(A$) : PRINT MID$(A$,N) : NEXT N
```

MID$(A$) gibt eine Unterdatenfolge wieder, die von der N-ten Stelle bis zum Ende der A$ (d. h. bis zum letzten Zeichen von A$) reicht. Die erste Stelle der Datenfolge ist Stelle 1, und die letzte wahrscheinliche Stelle der Datenfolge ist Stelle 255.

Sehr häufig kommt es vor, daß man nur das N-te Zeichen aus einer Datenfolge benötigt. Das läßt sich machen, indem man MID$ mit drei Argumenten abfordert, nämlich MID$(A$,N,1). Das dritte Argument gibt die Zahl der von Zeichen N ab wiederzugebenden Zeichen an.

Beispiel:

```
FOR N=1 TO LEN(A$) : PRINT MID$(A$,N,1), MID$(A$,N,2) : NEXT N
G              GO
O              OO
O              OD
D              D
               M
M              MO
O              OR
R              RN
N              NI
I              IN
N              NG
G              G
```

Weitere Details zur Arbeitsweise von LEFT$, RIGHT$ und MID$ finden sich in Kapitel 5.

Datenfolgen können auch verkettet (zusammengestellt oder verbunden) werden, indem man die (+)-Taste drückt. Versuchen Sie folgendes:

```
B$ = A$ + "   " + "BILL"
PRINT B$
GOOD MORNING BILL
```

Die Verkettung ist besonders nützlich, wenn Sie einen Teil der Datenfolge herausnehmen möchten und ihn mit kleinen Veränderungen wieder hineinsetzen wollen, z. B.:

```
C$ = RIGHT$(B$,=) + "-" + LEFT$(B$,4) + "-" + MID$(B$,6,7)
PRINT C$
BILL-GOOD-MORNING
```

Manchmal ist es wünschenswert, eine Zahl in ihre Datenfolgendarstellung umzuwandeln und umgekehrt. Mit Hilfe der Operationen VAL und STR$ werden diese Aufgaben erfüllt. Probieren Sie jetzt aus:

```
STRING$ = "567.8"
PRINT VAL(STRING$)
567.8

STRING$ = STR$(3.1415)
PRINT STRING$, LEFT$(STRING$,5)
3.1415          3.141
```

Mit Hilfe der STR$-Operationen lassen sich Zahlen in eine bestimmte Struktur für Eingabe oder Ausgabe verändern. Sie können eine Zahl zu einer Datenfolge machen und dann LEFT$, RIGHT$, MID$ und die Verkettung verwenden, um die Zahl in gewünschter Weise zu restrukturieren.

Im folgenden Beispiel wird gezeigt, wie man Datenfolgenoperationen zur Strukturierung des numerischen Output verwenden kann.

```
1ØØ INPUT "TYPE ANY NUMBER : ";X
11Ø PRINT : REM SKIP A LINE
12Ø PRINT "AFTER CONVERSION TO REAL PRECISION,"
13Ø INPUT "HOW MANY DIGITS TO RIGHT OF DECIMAL?"; D
14Ø GOSUB 1ØØØ
15Ø PRINT "*** " : REM SEPARATOR
16Ø GOTO 1ØØ
1ØØØ X$ = STR$(X) : REM CONVERT INPUT TO STRING
1Ø1Ø REM FIND POSITION OF E, IF IT EXISTS
1Ø2Ø FOR I = 1 TO LEN(X$)
1Ø3Ø IF MID$(X$,I,1)<>  "E" THEN NEXT I
1Ø4Ø REM I IS NOW AT EXPONENT PORTION (OR END)
1Ø5Ø REM FIND POSITION OF DECIMAL, IF IT EXISTS
1Ø6Ø FOR J = 1 TO I-1
1Ø7Ø IF MID$(X$,J,1)<> "." THEN NEXT J
1Ø8Ø REM J IS NOW AT DECIMAL (OR END OF NUMBER PORTION)
1Ø9Ø REM DO D DIGITS EXIST TO RIGHT OF DECIMAL?
11ØØ IF J+D    = I-1 THEN N = J+D :  GOTO 113Ø : REM YES
111Ø N = I-1 : REM NO, SO PRINT ALL DIGITS
112Ø REM PRINT NUMBER POSITION AND EXPONENT PORTION
113Ø PRINT LEFT$(X$,N) + MID$(X$,I)
114Ø RETURN
```

Beim o. g. Programm wird ein Unterprogramm, das auf Zeile 1ØØØ beginnt und eine vorher festgelegte reale verstümmelte Variable X in nichtgerundetem Zustand bis D Stellen nach dem Komma ausdrucken soll. Die Variablen X$, I und J werden in dem Unterprogramm als lokale Variablen verwendet.

In Zeile 1ØØØ wird die reale Variable in die Datenfolgenvariable X$. In Zeile 1Ø2Ø und 1Ø3Ø wird die Datenfolge geprüft, ob ein E vorhanden ist. I wird auf die Stelle von E gesetzt oder für LEN(X$)+1 eingegeben, wenn kein E vorhanden ist. In Zeile 1Ø6Ø und 1Ø7Ø wird das Komma gesucht. J wird auf die Stelle des Kommas oder auf I-1 gesetzt, wenn kein Komma vorhanden ist.

In der Zeile 11ØØ wird geprüft, ob es mindestens D Stellen nach dem Komma gibt. Wenn es sie gibt, muß der Zahlenteil der Datenfolge auf die Länge J+D beschnitten werden, d. h. auf D Stellen nach J, dem Komma. Die Variable N wird auf diese Länge gesetzt. Sollten weniger als D Stellen hinter dem Komma vorhanden sein, wird der gesamte Zahlenteil genutzt. Auf Zeile 111Ø wird die Variable auf diese Länge (I-1) gesetzt. Schließlich wird auf Zeile 113Ø die Variable X als die Verkettung zweier Unterdatenfolgen ausgedruckt. LEFT$(X$, N) gibt die bedeutsamen Stellen des Zahlenteils wieder und MID$(X$, I) gibt den Exponententeil wieder, soweit es ihn gibt.

STR$ kann auch genutzt werden, um leicht herauszufinden, wieviel Stellen für eine Zahl notwendig sein werden, z. B.:

```
PRINT LEN(STR$(33333.157))
9
```

Sie können die VAL-Operation anwenden, um die Zahlenwerte 5.36 und 5.1 aus der Frage WHAT IST THE VOLUME OF A CYLINDER OF RADIUS 5.36 FEET AND HEIGHT 5,1 FEET? (Wie groß ist das Volumen eines Zylinders mit den Maßen r = 5.36 Fuß und h = 5.1 Fuß?) entnehmen. Weitere Informationen über diese Operationen und CHR$ und ASC sind in Kapitel 5 zu finden.

Das nachstehend aufgeführte Programm sortiert eine Liste von Datenfolgen und druckt sie in alphabetischer Reihenfolge aus. Das Programm ähnelt sehr dem schon oben erläuterten zum Ordnen von Zahlen.

```
1ØØ DIN A$(15)
11Ø FOR I = 1 TO 15 : READ A$(I) : NEXT I
12Ø F = Ø : I = 1
13Ø IF A$(I) < = A$(I+1) THEN GOTO 18Ø
14Ø T$ = A$(I+1)
15Ø A$(I+1) = A$(I)
16Ø A$(I) = T$
17Ø F = 1
18Ø I = I+1 : IF I< = 15 THEN GOTO 13Ø
19Ø IF F = ; THEN GOTO 12Ø
```

```
200 FOR I = 1 TO 15 : PRINT A$(I) : NEXT I
220 DATA APPLE, DOG, CAT, RANDOM, COMPUTER, BASIC
230 DATA MONDAY, "*** ANSWER***", "FOO: "
240 DATA COMPUTER, FOO, ELP, MILWAUKEE, SEATTLE, ALBUQUERQUE
```

Weitere Beispiele für graphische farbige Darstellungen

In zwei Beispielen haben wir bereits erklärt, wie der APPLE II farbige graphische Darstellungen und Text produzieren kann. Im GR-Betrieb zeigt der APPLE bis zu 1600 kleine Quadrate auf einem 40 x 40 großen Gitter in einer von 16 Farben an. Er bietet auch 4 Zeilen für Text am unteren Rand des Bildschirms. Die waagerechte oder X-Achse ist normal, wobei 0 die Position links außen und 39 die Position rechts außen ist. Die senkrechte oder Y-Achse ist nicht normal insofern, da sie umgekehrt ist, denn 0 ist die oberste und 39 die unterste Position.

```
10 GR : REM INITIALIZE COLOR GRAPHICS;
   SET 40 X 40 TO BLACK.
   SET TEXT WINDOW TO 4 LINES AT BOTTOM
20 HOME : REM CLEAR ALL TEXT AT BOTTOM
30 COLOR = 1 : PLOT 0,0 : REM MAGENTA SQUARE AT 0,0
40 LIST 30 : GOSUB 1000
50 COLOR = 2 : PLOT 39,0 : REM BLUE SQUARE AT X=39, Y=0
60 HOME : LIST 50 : GOSUB 1000
70 COLOR = 12 : PLOT 0,39 : REM GREEN SQUARE AT X=0, Y=39
80 HOME : LIST 70 : GOSUB 1000
90 COLOR = 9 : PLOT 39,39 : REM ORANGE SQUARE AT X=39, Y=39
100 HOME : LIST 90 : GOSUB 1000
110 COLOR = 13 : PLOT 19,19 : REM YELLOW SQUARE AT CENTER OF SCREEN
120 HOME : LIST 110 : GOSUB 1000
130 HOME : PRINT "PLOT YOUR OWN POINTS"
140 PRINT "REMEMBER, X & Y MUST BE > = 0 & < = 39"
150 INPUT "ENTER X,Y : ";X,Y
160 COLOR = 8 : PLOT X,Y : REM BROWN SQUARES
170 PRINT "TYPE 'CTRL C' AND PRESS RETURN TO STOP"
```

```
180 GOTO 150
1000 PRINT "*** HIT ANY KEY TO CONTINUE *** "; : GET A$ : RETURN
```

Nachdem Sie das Programm eingetastet haben, lassen Sie es auflisten und untersuchen Sie es auf Druckfehler. Sie möchten es vielleicht auf einem Kassettentonband speichern? Lassen Sie jetzt das Programm ablaufen, indem Sie RUN eingeben.

Der GR-Befehl sorgt dafür, daß der APPLE auf den Farbbetrieb für graphische Darstellungen umschaltet . Durch COLOR geben Sie die Farbe des als nächstes zu druckenden Punktes ein. Diese Farbe bleibt, bis Sie durch COLOR eine andere Farbe wählen. Beispielsweise bleibt die Farbe in Zeile 160 unverändert, egal, wieviel Punkte gedruckt werden. Der Wert des Ausdrucks nach COLOR muß im Bereich von 0 bis 255 liegen, ansonsten erhalten Sie eine Fehlermeldung. Doch es gibt nur 16 verschiedene Farben, die gewöhnlich von 0 bis 16 numeriert sind.

Verändern Sie das Programm, indem Sie die Zeilen 150 und 160 umschreiben, so daß sie folgendes Aussehen haben:

```
150 INPUT "ENTER X, Y, COLOR :    "; X, Y, Z
160 COLOR = Z : PLOT X,Y
```

Lassen Sie jetzt das Programm laufen, und Sie können Ihre eigenen Farben und Punkte setzen. Wir werden gleich das ganze Farbspektrum des APPLE demonstrieren. Der Befehl PLOT X,Y läßt ein kleines Quadrat in der Farbe entstehen, die durch die letzte COLOR-Anweisung festgelegt wurde. Das Quadrat steht auf der durch X und Y definierten Stelle. Denken Sie daran, daß X und Y eine Zahl aus dem Bereich 0 bis 39 sein muß.

Die GET-Anweisung in Zeile 1000 gleicht einer INPUT-Anweisung. Der Computer wartet dann darauf, daß ein einziges Zeichen auf der Tastatur eingegeben wird, und ordnet dieses Zeichen der Variablen zu, die dem GET folgt. Es ist <u>nicht</u> notwendig, die RETURN-Taste zu bedienen. In Zeile 1000 wird das GET A$ nur benutzt, um das Programm solange zu unterbrechen, bis irgendeine Taste gedrückt wird.

Denken Sie daran, daß, wenn Sie vom Farbbetrieb für graphische Darstellungen auf Textbetrieb umschalten, TEXT eingeben müssen und dann die RETURN-Taste betätigen. Das APPLE-

SOFT-Kennungszeichen wird dann wi eder erscheinen.

Geben Sie das folgende Programm ein und lassen Sie es ablaufen, um das Farbspektrum des APPLE anzeigen zu lassen. (Beachten Sie: erst NEW eingeben!)

```
1Ø GR : HOME
2Ø FOR I = Ø TO 31
3Ø COLOR = I/2
4Ø VLIN Ø,39 AT I
5Ø NEXT I
6Ø FOR I = Ø TO 14 STEP 2 : PRINT TAB(I*2 + 1); I; NEXT I
7Ø PRINT
8Ø FOR I = 1 TO 15 STEP 2 : PRINT TAB(I*2 + 1); I; NEXT I
9Ø PRINT : PRINT "STANDARD APPLE COLOR BARS";
```

Die Farbstreifen werden doppelt so breit wie normal angezeigt. Ganz links befindet sich Schwarz, das durch COLOR=Ø gesetzt wird. Ganz rechts befindet sich Weiß, das durch COLOR=15 gesetzt wird. Wenn die Helligkeit ihres Fernsehgerätes richtig eingestellt ist, müßte der zweite Farbstreifen von links Magenta-Rot (rötliches Purpur), wie es durch COLOR=1 gesetzt wird, sein, und der dritte (COLOR=2) müßte sich in dunklem Blau zeigen. Stellen Sie Ihr Fernsehgerät mit dem Helligkeitsregler auf diese Farben ein. Die Farbtöne in Europa können unterschiedlich sein.

Im letzten Programm wurde ein Befehl in der Form VLIN Y1, Y2 AT X verwendet (Zeile 4Ø). Durch diesen Befehl wird eine senkrechte Linie von der Y-Koordinate Y1 zur Y-Koordinate Y2 beginnend in der Waagerechten bei Punkt X gezogen. Y1, Y2 und X müssen sich auf Werte im Bereich von Ø bis 39 beschränken. Y2 kann größer als, gleich oder kleiner als Y1 sein. Der Befehl HLIN X1, X2 AT Y ist dem VLIN-Befehl ähnlich, nur mit dem Unterschied, daß dadurch eine waagerechte Linie gezogen wird.

<u>Beachte:</u> Der APPLE zieht ebensoleicht eine ganze Linie, wie ein einzelner Punkt gedruckt wird!

Farbige graphische Darstellungen bei hoher Auflösung

Nun haben Sie den Farbbetrieb des APPLE bei niedriger Auflösung kennengelernt und werden leicht die Besonderheiten bei den graphischen Farbdarstellungen bei hoher Auflösung verstehen. Die Befehle ähneln sich. Gewöhnlich wird nur ein H (für hohe Auflösung) davorgesetzt. Beispielsweise bewirkt der Befehl HGR das Umschalten auf den hochauflösenden Farbbetrieb für graphische Darstellungen, das Freimachen des hochauflösenden Bildes bei gleichzeitiger Einstellung des schwarzen Untergrundes und die Fixierung der vier Zeilen für Text am unteren Rand des Bildes. Bei dieser Betriebsart setzen Sie Punkte in einem Gitter, das über 28Ø x- Positionen und 16Ø y-Positionen verfügt. In diesem Bereich lassen sich viel mehr Details darstellen, als in dem 4Ø x 4Ø großen Gitter der niedrigen Auflösung. Wenn Sie hier TEXT eingeben, dann wird auch auf normalen Textbetrieb umgeschaltet.

Zusätzlich zu dem oben erwähnten HGR-Bild gibt es noch ein zweites HGR-Bild, das Sie benutzen können, wenn Ihr APPLE mindestens 24 Kbyte hat. Die Betriebsweise für graphische Darstellungen der "zweiten Speicherseite" wird durch das Kommando HGR 2 eingeschaltet. Das Kommando macht den ganzen Bildschirm frei, wodurch Sie auf einer Fläche von 28Ø(x-Positionen) X 192(y-Positionen) Punkte setzen können. Bei dieser Betriebsart bleibt unten kein Rand für Text. Auch hier müssen Sie TEXT eingeben, wenn Sie Ihr Programm sehen möchten.

Das hört sich gut an, nicht wahr? Das ist es, aber Sie müssen etwas für die neue Fähigkeit opfern. Hierbei stehen weniger Farben zur Wahl. Die farbige Darstellung eines Graphs oder einer Abbildung wird durch das Kommando HCOLOR = N eingegeben, wobei N eine Zahl von Ø (schwarz) bis 7 (weiß) ist. Siehe auch Kapitel 8, wo eine vollständige Liste der zur Verfügung stehenden Farben enthalten ist. Aufgrund der Konstruktion der Farbfernsehgeräte verändern sich diese Farben von Gerät zu Gerät und von einerabgedruckten Linie zur anderen.

Schließlich gibt es noch eine einfache Instruktion für alle Arten des Druckens in den graphischen Darstellungen bei hoher Auflösung. Um das ganze in Aktion zu sehen, geben Sie ein:

```
HCOLOR = 3
HGR
HPLOT 13Ø, 1ØØ
```

Durch das letzte Kommando wird ein hochauflösender Punkt abgebildet, der in der Farbe, die Sie durch HCOLOR vorgegeben haben, an der Stelle x=13Ø, y=1ØØ entsteht. Wie bei den graphischen Darstellungen bei niedriger Auflösung befindet sich x=Ø an der linken Randbegrenzung des Bildschirms und y=Ø an der oberen Randbegrenzung. Bei höheren x-Werten rückt der Punkt nach rechts, bei höheren y-Werten rückt er nach unten. Der Höchstwert von x beläuft sich auf 279. Der Höchstwert von y ist 191. (bei dem HGR-Betrieb, wo Abbildungen und Text angezeigt werden, reichen die sichtbaren y-Werte nur bis y=159.)

Geben Sie jetzt ein: HPLOT 2Ø, 15 TO 145, 8Ø
Wie von einer unsichtbaren Hand gezogen wird eine weiße Linie vom Punkt x=2Ø, y=15 zum Punkt x=145, y=80 gezeichnet. HPLOT erlaubt Ihnen, Linien zwischen beliebigen Punkten auf dem Bild zu ziehen - sei es nun in horizontaler oder vertikaler Richtung oder im Winkel. Möchten Sie gern noch eine andere Linie an die vorherige anschließen? Nun, dann geben Sie ein: HPLOT TO 12, 8Ø.

Bei dieser Kommandoform wird der Endpunkt der vorher gedruckten Abbildung als der Anfangspunkt für die neue genommen. Dabei wird auch die Farbe von diesem Punkt übernommen (selbst wenn Sie bereits ein neues HCOLOR inzwischen eingegeben haben sollten). Sie können diese Befehle sogar in einer Anweisung "kettenförmig aneinanderreihen". Versuchen Sie jetzt folgendes:

```
HPLOT Ø,Ø TO 279,Ø TO 279,159 TO Ø, 159 TO Ø,Ø
```

Sie müßten jetzt ein weiß umrandetes Bild auf dem Schirm haben!

Das hier ist ein Programm, das hübsche Wellenmuster auf Ihrem Bildschirm sichtbar macht.

```
8Ø HOME : REM CLEAR THE TEXT AREA
1ØØ VTAB 24 : REM MOYE CURSOR TO BOTTOM LINE
12Ø HGR : REM SET HIGH-RESOLUTION GRAPHICS MODE
14Ø A = RND(1) * 279 : REM PICK AN X FOR "CENTER"
15Ø B = RND(1) * 159 : REM PICK A Y FOR "CENTER"
16Ø I% = RND (1) * 4) + 2 : REM PICK A STEP SIZE
2ØØ HTAB 15 : PRINT "STEPPING BY  "; I%;
22Ø FOR X = Ø TO 278 STEP I% : REM  STEP THRU X VALUES
24Ø FOR S = Ø To 1  : REM  2 LINES, FROM X AND X+1
```

```
260 HCOLOR = 3*S  :  REM  FIRST LINE BLACK, NEXT WHITE
280 REM DRAW LINE THROUGH "CENTER" TO OPPOSITE SIDE
300 HPLOT X+S,0 TO A,B TO 279-X-S,159
320 NEXT S, X
340 FOR Y = 0 TO 158 STEP I%  :  REM  STEP THRU Y VALUES
360 FOR S = 0 TO 1  :  REM  LINES FROM Y AND Y+1
380 HCOLOR = 3*S  :  REM  FIRST LINE BLACK? NEXT WHITE
400 REM  DRAW TROUGH "CENTER" TO OPPOSITE SIDE
420 HPLOT 279,Y+S TO A,B TO 0,159-Y-S
440 NEXT S, Y
460 FOR PAUSE = 1 TO 1500  :  NEXT PAUSE  :  REM  DELAY
480 GOTO 120  :  REM  DRAW  A NEW  PATTERN
```

Das ist ein ziemlich langes Programm. Geben Sie es sorgsam ein und lassen Sie es in Teilabschnitten aufLISTen &z.B.LIST 0,320),um Ihre Eingabe zu prüfen. Wir haben zwischen einigen Zeilen einen Zwischenraum gelassen, damit man das Programm einfacher lesen kann. Beim Auflisten werden keine Zwischeräume gelassen. Wenn Sie sicher sind, daß das Programm richtig ist, lassen Sie es ablaufen.

VTAB und HTAB sind Kommandos, die den Cursor bewegen und benutzt werden, um ein Zeichen an einer vormarkierten Stelle auf dem Textbildschirm schreiben zu lassen. VTAB 1 bewegt den Läufer in die oberste Zeile,VTAB 24 in die unterste Zeile. HTAB 1 bewegt den Läufer in die linke Randposition der laufenden Zeile, HTAB 40 in die rechte Randposition. In einer PRINT-Anweisung wie in Zeile 200 des Programms werden Sie ein Schlußsemikolon benötigen, um den Zeilenvorschub zu verhindern, der Ihre Nachricht verschiebt.

Die Operation RND(N), bei der N eine beliebige positive Zahl ist, ergibt eine zufällige Zahl im Bereich von 0 bis .999999999. (Siehe auch Kapitel 10 zur vollständigen Erläuterung von RND). So wird in Zeile 180 der ganzen Variable I% eine zufällige Zahl von 2 bis 5 (eine Zahl wird immer abgerundet, wenn sie in eine ganze Zahl verwandelt wird) zugeordnet. Die STEP-Größe in einer FOR...NEXT-Schleife muß keine ganze Zahl sein, aber die Ergebnisse lassen sich für einen ganzzahligen STEP leichter voraussagen.

Wie Sie in Zeile 32Ø und 44Ø sehen können, können mehrere FOR-Anweisungen stehen für eine NEXT-Anweisung. Denken Sie aber daran, daß Sie die NEXT-Variablen in der richtigen Reihenfolge auflisten, um überschnittene Schleifen zu vermeiden!

Die Zeile 46Ø enthält lediglich eine "Verzögerungsschleife", die es Ihnen ermöglicht, ein Muster einen Augenblick bewundern zu können, ehe das nächste beginnt. Jedesmal schickt Zeile 48Ø den Computer zum HGR-Befehl auf Zeile 12Ø zurück, wodurch das Bild gelöscht wird und der Freiraum für das nächste entsteht.

Wenn Sie wieder neu programmieren wollen, dann stoppen Sie das Musterschreiben mit Hilfe von ctrl C und geben dann ein: TEXT

Haben Sie bereits eine Vorstellung, wie Sie das Programm ändern können? Nachdem Sie diese Version auf Ihrer Kassette oder Platte geSAVET haben, versuchen Sie,den HCOLOR-Wert wahlfrei veränderbar zu machen. Versuchen Sie erst weiße und dann schwarze Linien zeichnen zu lassen, oder vielleicht nur weiße?

Also, dann viel Freude beim Programmieren!

KAPITEL 2

Definitionen

Definitionen und Abkürzungen der Syntax

Bei den folgenden Definitionen werden Metasymbole , wie die Zeichen { und \ benutzt, um Strukturen und Beziehungen in APPLESOFT eindeutig festzulegen. Die Metasymbole gehören nicht zu APPLESOFT. Zusätzlich zu den Metasymbolen gibt es noch da: spezielle Symbol : = , welches den Anfang einer vollständigen Definition oder Teildefinition des links vom : = stehenden Terminus kennzeichnet.

| : = Metasymbol zur Trennung von Alternativen (Beachte, daß eine Position getrennt nach Alternativen, definiert werden kann).

[ ] : = Metasymbol zur Einklammerung von wahlfreien Ausdrücken.

{ } : = Metasymbol zur Einklammerung von Ausdrücken, die sich wiederholen können.

\ : = Metasymbol zur Einklammerung von Ausdrücken, deren Wert benutzt wird. Der Wert von x wird \ x\ geschrieben.

~ : = Metasymbol zur Kennzeichnung eines benötigten Raums.

Metasymbol

: = ||[|]|{|}|\|~

Kleinbuchstabe

: = a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z

Metymbol

: = Kleinbuchstabe

Ziffer

: = 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ø

Metaname

: = { Metasymbol } [Ziffer]

Metasymbol

: = eine einfache Ziffer verkettet mit einem Metanamen

APPLESOFT-II-Spezialsymbol

: = Sonderzeichen

Sonderzeichen

: = ! | # | % | & | ' | (|) | $ | : | = | - | @ | + | ; | ? | / | . | < | * | , |] | ^ |

Steuerungszeichen (Zeichen, die bei gleichzeitiger Betätigung der CTRL-Taste eingegeben werden) und die Null sind ebenfalls Sonderzeichen. APPLESOFT verwendet die eckige Klammer nach links ( ]) nur als Ladesymbol. In dieser Beschreibung wird sie als Metasymbol benutzt.

Buchstabe

: = A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

Zeichen

: = Buchstabe | Ziffer | Sonderzeichen

Alphanumerisches Zeichen

: = Buchstabe | Ziffer

Name

: = Buchstabe [{ Buchstabe | Ziffer }]

Ein Name darf bis zu 238 Zeichen lang sein. Bei der Unterscheidung von Namen werden bei APPLESOFT alle beliebigen alphanumerischen Zeichen außer den ersten beiden Zeichen nicht beachtet. Im APPLESOFT werden keine Unterschiede zwischen den Namen GOOD4LITTLE und GOLDRUSH gemacht. Im nicht berücksichtigen Namenteil dürfen keine Sonderzeichen, das Anführungszeichen (") oder eines der Tabuwörter von APPLESOFT enthalten sein. (Siehe dazu Anlage A, in der eine Aufstellung der Tabuwörter und Anmerkungen zu den Ausnahmen

zu dieser Regel gegeben werden.)

Ganze Zahl

: = [+ | -] {Ziffer}

Ganze Zahlen müssen im Bereich -32767 bis+32767 liegen. Bei der Umwandlung von nicht-ganzen Zahlen in ganze Zahlen wird APPLESOFT in der Regel die nicht ganze Zahl zu der nächstkleineren ganzen Zahl verändern. Das trifft aber nicht zu, wenn sich die nicht-ganze Zahl der nächstgrößeren ganzen Zahl nähert.

Beispiel:

A% = 123, 999 999 959 999	B% = 123. 999 999 96
PRINT A%	PRINT B%
123	124
C% = 12345. 999 995 999	D% = 12345. 999 996
PRINT C%	PRINT D%
12345	12346

(Zwischenräume nur zur Erleichterung des Lesens!)

Eine ganze Bereichszahl belegt 2 Byte (16 bit) im Speicher.

Name für eine ganze Variable

: = Name%

Eine reelle Zahl kann als eine ganze Variable gespeichert werden, jedoch wird vorher durch APPELSOFT die Umwandlung von der reellen Zahl in die ganze Zahl vorgenommen.

Reelle Zahl:

: =[+ | -]{Ziffer} [.{Ziffer}] [E [+ | -] Ziffer [Ziffer]]
: =[+ | -][{Ziffer}] .[{Ziffer}] [E [+ | -] Ziffer [Ziffer]]

Der Buchstabe E, wie er in der Schreibweise von reellen Zahlen verwendet wird, steht für "Exponent". Es ist das Kurzzeichen für * 1Ø ^ . Dabei wird Zehn in die Potenz erhoben, wie sie von der Zahl hinter dem E fest-

gelegt wird. Das Ergebnis wird mit der vor dem E stehenden Zahl multipliziert.
Für APPLESOFT müssen die reellen Zahlen in dem Bereich von - 1E38 bis + 1E38 liegen.
Anderenfalls wird auf dem Bildschirm die Fehlermeldung ?OVERFLOW ERROR angezeigt.
Bei der Addition und der Subtraktion können Zahlen erzeugt werden bis zu einer Größe von 1.7E 38, ohne daß eine Fehlermeldung angezeigt wird.

Eine reelle Zahl, deren absoluter Wert unter 2.9388E-39 liegt, wird von Applesoft zu Null umgerechnet.
Bei APPLESOFT werden die nachstehend aufgeführten Druckzeichen als reelle Zahlen betrachtet, wenn sie allein dargestellt sind. Diese werden dann als Null behandelt.

. +. -. .E +.E -.E .E+ .E- +.E- +.E+ -.E+ -.E-

Deshalb sind die Bereichselemente M (.) und M (Ø) identisch. Außer den abgekürzten reellen Zahlen, die sich in der Liste oben befinden, werden auch noch die nachfolgenden Druckzeichen als reelle Zahlen angesehen und als Null behandelt, wenn sie als numerische Antworten für das INPUT oder als numerische Elemente in DATA benutzt werden:

+ - E +E -E space E+ E- +E+ +E- -E+ -E-

In der GET-Anweisung werden alle die aus einem Zeichen bestehenden reellen Zahlen, die in den Listen oben zu finden sind, wie Null behandelt.

Wenn eine reelle Zahl geschrieben wird, wird durch APPLESOFT maximal eine neunstellige Zahl gebildet (Ausnahmen siehe weiter unten), einschließlich des Exponenten (sofern vorhanden). Jede weitere Stelle wird gerundet. Nullen, die der ersten von Null verschiedenen Ziffer vor dem Komma vorausgehen, werden nicht geschrieben. Nullen, die der letzten von Null verschiedenen Ziffer nach dem Komma folgen, werden ebenfalls nicht geschrieben.
Gibt es keine von Null verschiedene Ziffer nach dem Komma, wird das Komma nicht geschrieben.

Achtung !

Das Runden kann recht eigentümlich sein:

PRINT 99 999 999.9

99 999 999.9

PRINT 99 999 999.9Ø
1ØØ ØØØ ØØØ

PRINT 11.111 111 45Ø ØØ
11.111 111 5

PRINT 11.111 111 451 9
11.111 111 4

(Zwischenräume nur zur Erleichterung des Lesens!)

Ist der absolute Wert einer reellen Zahl größer als oder gleich .Ø1 und kleiner als 999 999 999.2, dann wird die reelle Zahl ohne Gleitkomma gedruckt, d. h., es wird kein Exponent angezeigt. In dem Bereich .Ø 1ØØ ØØØ ØØØ 5bis .Ø 999 999 999 werden die reellen Zahlen mit bis zu zehn Stellen nach dem Komma angegeben. Das ist die einzige Ausnahme zur angegebenen Begrenzung auf neun gedruckte Stellen, einschließlich des Exponenten.

Der Versuch, eine Zahl mit über 38 Stellen zu benutzen, wie z. B. 211.1111111111111111111111111111111111111, bringt die Meldung:

?OVERFLOW ERROR

Sie erfolgt auch dann, wenn die Zahl eindeutig im zugelassenen Bereich - 1E38 bis +1E38 liegt. Das gilt auch dann, wenn die meisten der Ziffern, aufeinanderfolgende Nullen sind, wie im Beispiel: 211.ØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØ. Die führenden Nullen werden jedoch nicht berücksichtigt. Ist die erste Ziffer eine 1 und die zweite kleiner als oder gleich 6, können Zahlen mit 39 Stellen benutzt werden, ohne daß die Fehlermeldung angezeigt wird.

Eine reelle Zahl belegt 5 Byte (4Ø bit) im Speicher.

Reeller Variablenname

: = Name

Arithmetische Veriable

: = avar

avar

: = Name | Name%

Alle einfachen Variablen belegen 7 Byte im Speicher, davon 2 Byte für den Namen und 5 Byte für den Wert der reellen oder ganzen Zahl.

Trennsymbol

: = ~ | (|) | = | - | + | ^ |>|< | /|*|, | ; | :

Ein Name braucht nicht von einem vorausgehenden oder folgenden Tabuwort durch eines dieser Trennsymbole getrennt werden.

Arithmetischer Operator

: = aop

aop

: = + | -| *| /|^

Arithmetisch-logischer Operator

: = alop

alop

: = AND| OR | =|>|<|<>|><|>≡| =>|<=| =<

Operator

: = op

op

: = aop | alop

Arithmetischer Ausdruck

: = aexpr

aexpr

: = avar| reelle | ganze

: = (aexpr)

Wenn die Klammern bis über 36 Ebenen hinaus verschachtelt werden,

wird ?OUT OF MEMORY ERROR angezeigt.

: =[+ | - | NOT] aexpr

Das monadisch NOT erscheint hier gemeinsam mit dem monadischen + und -.

: = aexpr op aexpr

Index

: = (aexpr[{, aexpr}])

Die maximale Länge beträgt 89, obwohl das in der Praxis durch den Umfang des vorhandenen Speichers begrenzt ist. aexpr muß positiv sein. Bei der Verwendung wird er in eine ganze Zahl umgewandelt.

avar

: = avar Index

aexpr

: = avar Index

Literal

: ="[{ Zeichen}]"

Gruppe

: = "[{ Zeichen}] "

Eine Gruppe belegt 1 Byte (8 bit) zur Speicherung der Länge, 2 Byte für den Stellenzeiger und 1 Byte pro Zeichen innerhalb der Gruppe.

: = "[{Zeichen}]return

Diese Gruppenform kann nur am Ende der Zeile erscheinen.

Nullgruppe

: = ''''

Gruppenvariablenname

: = Name $

Gruppenvariable

: = svar

svar

: = Name $ | Name$ Index

Der Stellenzeiger und der Variablenname belegen jeweils 2 Byte. Die Länge und jedes Gruppenzeichen belegen je 1 Byte.

Gruppenoperator

: = sop

sop

: = +

Gruppenausdruck

: = sexpr

sexpr

: = svar | Gruppe

: = sexpr sop sexpr

Logischer Gruppenoperator

: = slop

slop

: = = | > | > = | => | < | <= | =< | <> | ><

aexpr

: = sexpr slop sexpr

Variable

: = var

var

: = avar | sexpr

Ausdruck

: = expr

expr

: = aexpr : sexpr

Ladesymbol

: =

Die eckige, nach rechts offene Klammer wird angezeigt, wenn APPLE-SOFT zur Aufnahme eine s weiteren Befehls bereit ist.

reset

: = ein Druck auf die mit "RESET" markierte Taste

esc

: = ein Druck auf die mit "ESC" markierte Taste

return

: = ein Druck auf die mit "RETURN" markierte Taste

ctrl

: = anhaltendes Drücken der mit "CTRL" markierten Taste bei der danach folgenden Buchstabentaste..

Zeilennummer

: = linenum

linenum

: = { Ziffer }

Die Zeilennummer muß im Bereich zwischen Ø bis 63999 liegen. Andernfalls wird ?SYNTAX ERROR angezeigt.

Zeile

: = linenum [{Anweisung:}] Anweisung return

Eine Zeile kann eine Länge bis zu 239 Zeichen haben, wobei alle Leerräume , die vom Benutzer eingegeben werden, mitgezählt werden. Zu den 239 Zeichen gehören nicht die von APPLESOFT zur besseren Strukturierung der Zeile hinzugefügten Leerräume.

Regeln für die Auswertung von Ausdrücken

Die Operatoren sind in senkrechter Anordnung,entsprechend der Reihenfolge ihrer Ausführung, beginnend mit der höchsten Priorität (Klammern) bis zur niedrigsten Priorität (OR),aufgelistet. Operatoren, die sich auf der gleichen Zeile befinden, gehören der gleichen Prioritätsstufe an und werden von links beginnend ausgeführt.

( )
+ - NOT monadische Operatoren
^
* /
+ -
> < >= <= => =< <> >< =
AND
OR

Umwandlung von Eingaben

Wenn eine ganze Zahl und eine reelle Zahl in einer Berechnung vorhanden sind, dann werden alle Zahlen in reelle Zahlen umgewandelt, bevor die Berechnung ausgeführt wird. Die Ergebnisse werden in die arithmetische Form (ganze oder reelle) der endgültigen Variable verwandelt, der sie zugeordnet werden. Bei Funktionen, die auf einen gegebenen arithmetischen Typ festgelegt sind, müssen die Argumente eines anderen Typs in den Typ verwandelt werden, für den sie definiert sind. Gruppen und arithmetische Eingaben können nicht gemischt werden. Sie können mit Hilfe der dafür vorgesehenen Funktionen in die jeweils andere Art umgewandelt werden.

Ausführungsweisen

imm Einige Anweisungen können im APPLESOFT sofort (immediate - imm) ausgeführt werden. Dabei muß die Anweisung ohne Zeilennummer eingegeben werden. Bei der Betätigung der RETURN-Taste wird die Anweisung sofort ausgeführt.

def Anweisungen, die im verzögerten Ausführungsbetrieb eingegeben werden, müssen mit einer Zeilennummer versehen sein. Bei der Betätigung der RETURN-Taste speichert der Computer die numerierte Zeile zwecks späterer Ausführung. Die Anweisungen bei verzögerter(deferred - def) Ausführung werden erst ausgeführt, wenn die Liste der Anweisungen, also das Programm, mit dem Kommando RUN abgeschlossen wird.

KAPITEL 3

Seite

System- und Programmbefehle

LOAD imm & def
SAVE imm & def

LOAD
SAVE

Hiermit wird ein Programm vom Kassettentonband geladen bzw. auf dem Kassettentonband festgehalten. Diese Befehle bewirken weder eine Bereitschaftsmeldung noch irgendein anderes Signal. Der Benutzer muß dafür sorgen, daß das Tonbandgerät richtig eingestellt ist (Wiedergabe oder Aufnahme), wenn der Befehl ausgeführt wird.
Wenn LOAD oder SAVE eingegeben wird, erfolgt keine Prüfung, ob der Recorder richtig eingestellt ist oder gar, ob überhaupt ein Tonbandgerät angeschlossen ist. Bei beiden Kommandos ertönt ein Ton ("biep"), um den Anfang und das Ende der Aufnahme bzw. der Wiedergabe anzuzeigen.

Die Programmausführung wird nach einer SAVE-Operation fortgesetzt, doch bei LOAD wird das laufende Programm gelöscht, wenn das Lesen neuer Informationen vom Tonband beginnt.

Allein reset kann LOAD oder SAVE unterbrechen.

Wenn die Tabuwörter LOAD oder SAVE als erste Zeichen eines Variablennamens verwendet werden, kann der durch das Tabuwort erteilte Befehl ausgeführt werden, bevor irgendeine Fehlermeldung

? SYNTAX ERROR

angezeigt wird. Eine Anweisung wie bspw.

SAVERING = 5

würde bewirken, daß das laufende Programm festgehalten werden soll. Sie können entweder auf den "Biep"-Ton (und die Fehlermeldung ?SYNTAX ERROR) warten oder reset betätigen.

Die Anweisung

LOADTOJOY = 47

hält das System in der Schwebe, während APPELESOFT das laufende Programm löscht und unendlich lange auf ein Programm vom Kassettentonband wartet. Nur wenn Sie die reset-Taste betätigen, können Sie wieder die Steuerung des Computers übernehmen.

NEW imm & def
NEW

Keine Parameter. Löscht das laufende Programm und alle Variablen.

RUN imm & def

RUN linenum

Bewirkt die Löschung aller Variablen, Anzeigen und aller Stacks.

Die Ausführung des Programms beginnt mit der angegebenen linenum, Zeilennummer. Ist diese nicht angegeben, setzt das RUN bei der Zeile im Programm an, die die niedrigste Zeilennummer trägt. Ist kein Programm gespeichert, bewirkt es, daß dem Nutzer die Steuerung freigegeben ist.

Bei der verzögerten Ausführung erscheint die Fehlermeldung

? UNDEF'D STATEMENT ERROR ,

wenn eine Zeilennummer angegeben ist, die nicht in dem Programm enthalten oder eine negative ist. Überschreitet die Zeilennummer +63999, dann erscheint auf der Anzeige die Fehlermeldung

? SYNTAX ERROR ,

wobei nicht angegeben wird, in welcher Zeile es zu dem Fehler gekommen ist.

Bei der unmittelbaren Ausführung werden andererseits die beiden genannten Fehlermeldungen zu

? UNDEF'D STATEMENT ERROR IN xxxx und

? SYNTAX ERROR IN xxxx ,

wobei xxxx die verschiedensten Zeilennummern, gewöhnlich über 65ØØØ, bezeichnen kann.

Wenn RUN in einem Sofortausführungsprogramm benutzt wird, werden jegliche Teile des Sofortausführungsprogramms, die danach ausgeführt werden sollten, nicht ausgeführt.

STOP imm & def
END imm & def
ctrl C nur imm
reset nur imm
CONT imm & def

STOP
END
ctrl C
reset
CONT

Geben Sie STOP ein, so wird die Ausführung eines Programms unterbrochen, und die Steuerung des Computers wird dem Nutzer freigegeben. Auf der Anzeige erscheint:

BREAK IN linenum

Das "linenum" ist dabei die Zeilennummer, in deren Anweisung die Unterbrechung, das STOP, ausgeführt wurde.

END läßt die Ausführung eines Programms beenden und räumt dem Nutzer die weitere Steuerung des Computers ein. Dabei wird keine Meldung angezeigt.

Mit ctrl C erzielt man den gleichen Effekt wie durch das Einfügen einer STOP-Anweisung unmittelbar nach der Anweisung, die gerade vom Computer ausgeführt wird. Man kann ctrl C benutzen, um eine Auflistung zu unterbrechen. Außerdem läßt es sich zur Unterbrechung eines INPUT benutzen, aber erst, nachdem das erste Zeichen eingegeben worden ist. Das INPUT wird erst dann unterbrochen, wenn return eingegeben wird.

Das reset stoppt jegliche APPLESOFT-Programme oder Befehle unbedingt und sofort. Das Programm geht dabei nicht verloren, aber einige Programmzeiger und -stacks werden gelöscht. Dieser Befehl bewirkt, daß Sie in dem Monitorprogramm des Systems bleiben, wie das durch das Bereitschaftszeichen des Monitors (*) angezeigt wird. Um zum APPLESOFT zurückzukehren, ohne das laufende Programm zu zerstören, müssen Sie ctrl C return eingeben.

Wurde die Programmausführung durch STOP, END oder ctrl C unterbrochen, dann kann man mit dem CONT-Befehl die Wiederaufnahme der Programmausführung bei der nächsten Anweisung, <u>nicht</u> beginnend bei der nächsten Zeilennummer, veranlassen. Es wird nichts gelöscht.

Wurde kein Programm unterbrochen, dann erzielt man mit CONT keinen Effekt. Nach der Nullstellung durch ctrl C return könnte die Programmausführung nicht ordentlich weiterlaufen, denn einige Programmzeiger und -stacks werden gelöscht worden sein.

Wurde eine INPUT-Anweisung durch ctrl C unterbrochen, dann wird der Versuch, die Programmausführung durch CONT wieder aufnehmen zu lassen, zur Fehlermeldung

? SYNTAX ERROR IN linenum

führen, wobei linenum die Zeilennummer ist, die die INPUT-Anweisung enthält.

Die Ausführung des CONT-Befehls wird zu der Fehlermeldung

? CAN'T CONTINUE ERROR

führen, wenn der Benutzer, nachdem die Programmausführung unterbrochen ist,

a) irgendeine Programmzeile verändert oder löscht.

b) irgendeine Operation versucht, die zu einer Fehlermeldung führt.

Allerdings können Programmvariablen verändert werden, wenn Sofortausführungsbefehle benutzt werden, so lange es nicht zu Fehlermeldungen kommt.

ACHTUNG!

Falls bei der verzögerten Ausführung DEL benutzt wird, werden die entsprechenden Zeilen gelöscht, und die Programmausführung unterbricht an dieser Stelle. Ein Versuch, mit Hilfe von CONT das Programm wieder in Gang zu bringen, würde die Fehlermeldung

? CAN'T CONTINUE ERROR

nach sich ziehen.

Wird CONT in einer Anweisung für verzögerte Ausführung benutzt, dann wird die Programmausführung bei dieser Anweisung unterbrochen, aber die Steuerung des Computers wird nicht für den Benutzer freigegeben. Der Benutzer kann die Steuerung des Computers wieder in seine Hand bekommen, wenn er einen ctrl-C-Befehl erteilt, doch ein Versuch, dann in der nächsten Anweisung, die Programmausführung wieder aufnehmen zu lassen durch ein CONT, würde nur bedeuten, daß der Computer wieder durch das unterbrochene Programm gesteuert würde.

TRACE imm & def
NOTRACE imm & def

TRACE
NOTRACE

Mit TRACE wird auf eine Fehlersuch-Betriebsart umgeschaltet, bei der die Zeilennummer einer jeden Anweisung bei deren Ausführung angezeigt wird. Wird das Programm auf dem Bildschirm angezeigt, dann können die TRACEs (Fehlerstellen) auf unerwartete Weise angezeigt werden oder überschrieben erscheinen. NOTRACE schaltet den TRACE-Fehlersuchbetrieb ab.

Wenn einaml TRACE gesetzt wurde, dann wird es nicht durch ein RUN, CLEAR, NEW, DEL oder reset abgeschaltet. Das Nullstellen mit ctrl B schaltet den TRACE-Betrieb ab (und löscht jedes gespeicherte Programm).

PEEK imm & def

PEEK (aexpr)

Durch PEEK wird der Inhalt des Bytes auf der Adresse \aexpr\ dezimal wiedergegeben. Anlage J enthält Beispiele für die Benutzung von PEEK.

POKE imm & def

POKE aexpr1, aexpr2

Mit POKE wird eine Menge von 8 bit, nämlich das binäre Äquivalent des Dezimalwertes von \aexpr2\ auf der Speicherstelle gespeichert, deren Adresse durch \aexpr1\ gegeben ist. Der Bereich von \aexpr2\ reicht von Ø bis 255, der von \aexpr1\ von -65535 bis 65535. Vor der Ausführung werden reelle Zahlen in ganze verwandelt. Werte, die außerhalb des Bereichs liegen, verursachen die Anzeige der Fehlermeldung:
? ILLEGAL QUANTITY ERROR

\aexpr2\ wird nur dann tatsächlich gespeichert, wenn die entsprechende Hardware (Speicher oder geeignetes OUTPUT-Gerät) für die von \aexpr1\ bestimmte Adresse vorhanden ist. \aexpr2\ wird nicht gespeichert, wenn nichtaufnehmende Adressen wie beispielsweise Monitor-ROM oder unbenutzte Input/Output-Eingänge belegt werden.

Im allgemeinen bedeutet das, daß \aexpr1\ im Bereich Ø bis max. liegt, wobei max. von der Speicherkapazität des Computers bestimmt wird. Beispielsweise beträgt max. beim APPLE II mit 16 K-byte 16384. Hat der APPLE II 32 K-byte Speicherkapazität, dann beträgt max. 32768, und bei einem APPLE II mit 48 K-byte liegt max. bei 49152.

Sehr viele Speicherstellen enthalten Informationen, die für den Betrieb des Computer-Systems notwendig sind. Ein POKE in diese Speicherstellen hinein kann den Betrieb des Systems oder Ihres Programms verändern oder gar dem APPLESOFT einen k.o. versetzen.

WAIT imm & def

WAIT aexpr1, aexpr2 [,aexpr3]

Mit dem WAIT erhält der Benutzer die Möglichkeit, eine bedingte Pause in das Programm einzufügen. Nur mit reset können Sie ein WAIT unterbrechen.

\aexpr1\ ist die Adresse der Speicherstelle; sie muß im Bereich -65535 bis 65535 liegen. Andernfalls wird die Fehlermeldung:
? ILLEGAL QUANTITIY ERROR

angezeigt. Praktisch wird \aexpr1\ durch den Bereich der Adressen beschränkt, die Speicherstellen ansprechen, auf denen gültige Speichergeräte existieren. Der Bereich erstreckt sich von Ø bis zum Maximalwert für HIMEM: in ihrem Computer. Unter HIMEM: und POKE finden sich weitere Einzelheiten. Äquivalente positive und negative Adressen können verwendet werden.

\aexpr2\ und \aexpr3\ müssen im Bereich Ø bis 255 dezimal sein.
Bei der Ausführung eines WAIT werden diese Werte in binäre Zahlen im Bereich von Ø bis 11111111 umgerechnet.

Wenn nur \aexpr1\ und \aexpr2\ spezifiziert sind, dann wird ein jedes der acht Bits im binären Inhalt der Stelle \aexpr1\ mit dem entsprechenden Bit im binären Äquivalent von \aexpr2\ geUNDet. Bei jedem Bit ergibt das eine Null, wenn nicht beide der entsprechenden Bits (1) sind. Ergeben sich aus diesem Prozeß acht Nullen, dann wird der Test wiederholt. Sollte ein Resultat ein von Null verschiedenes sein (was bedeutet, daß mindest ein auf 1 gesetztes Bit in \aexpr2\ mit einem entsprechenden auf 1 gesetzten Bit auf der Speicherstelle \aexpr1\ zusammengefügt wurde), ist das WAIT vollendet, und das APPLESOFT-Programm wird mit der nächsten Anweisung weiter ausgeführt.
WAIT aexpr1, 7
führt dazu, daß das Programm so lange unterbrochen wird, bis zumindest eines der drei rechtsbündigen Bits auf Speicherstelle \aexpr1\ ein Bit (1) ist.
WAIT aexpr1, Ø

löst eine immerwährende Pause des Programmes aus.

Wenn alle drei Parameter spezifiziert sind, dann läuft das WAIT folgendermaßen ab. Zunächst wird jedes Bit im binären Inhalt der Stelle \aexpr1\ mit dem entsprechenden Bit in dem binären Äquivalent von \aexpr3\ geXODERt. Ein Bit (1) in \aexpr3\ ergibt die Umkehrung des entsprechenden Bit auf der Stelle \aexpr1\ (1 wird zu Ø und Ø wird zu 1). Ein Bit (Ø) in \aexpr3\ ergibt, daß das entsprechende Bit auf der Stelle \aexpr1\ gleich bleibt. Sollte \aexpr3\ gerade Ø sein, dann geschieht nichts durch den XODER-Teil (XODER ist exklusives ODER).

Als nächstes wird dann jedes Ergebnis mit dem entsprechenden Bit in dem binären Äquivalent von \aexpr2\ geUNDet. Ergeben sich daraus abschließend acht Nullen, wird der

Test wiederholt. Sollte ein von Null verschiedenes Ergebnis entstehen, dann ist das WAIT abgeschlossen, und der APPLESOFT-Programmablauf setzt sich mit der nächsten Anweisung fort.

Hier noch eine andere Betrachtungsweise für das WAIT: Das Ziel besteht darin, den Inhalt der Stelle \aexpr1\ zu prüfen, ob irgendeines aus einer bestimmten Menge Bits ein Bit (1) ist oder irgendein anderes aus einer anderen Menge ein Bit (Ø) ist. Jedes der acht Bits im binären Äquivalent von \aexpr2\ gibt an, ob Sie an dem entsprechenden Bit auf Stelle \aexpr1\ interessiert sind oder nicht. 1 bedeutet ja, Ø bedeutet nein. Jedes der acht Bit in dem binären Äquivalent von \aexpr3\ gibt an, welchen Zustand Sie für das entsprechende Bit auf Stelle \aexpr1\ erwarten. 1 bedeutet, daß das Bit eine Ø sein muß; Ø bedeutet, daß das Bit eine 1 sein muß. Wenn <u>irgendeines</u> der Bits, für das Sie sich interessieren (durch Angabe von 1 in dem entsprechenden Bit von \aexpr2\), mit dem Zustand übereinstimmt, den Sie für das Bit festgelegt haben (durch das entsprechende Bit von \aexpr3\ ), dann ist das WAIT abgeschlossen. Wenn \aexpr3\ ausgelassen wird, beträgt der Fehlerwert Ø.

Beispiel:

WAIT aexpr1, 255, Ø	bedeutet eine Pause, bis mindestens eines der acht Bit auf Stelle \aexpr1\ eine 1 ist.
WAIT aexpr1, 255	identisch mit dem o.g.
WAIT aexpr1, 255, 255	bedeutet eine Pause, bis mindestens eines der acht Bits auf Stelle \aexpr1\ eine Ø ist.
WAIT aexpr1, 1, 1	bedeutet eine Pause, bis das rechtsbündige Bit auf Stelle \aexpr1\ eine Ø ist, unabhängig davon, welchen Zustand die anderen haben.
WAIT aexpr1, 3, 2	bedeutet eine Pause, bis entweder das rechtsbündige Bit auf Stelle \aexpr1\ eine 1 ist, oder das vor dem rechtsbündigen stehende Bit eine Ø ist oder bis beide Bedingungen erfüllt sind.

Dieses Programm pausiert so lange, bis Sie irgendein Zeichen eingeben, dessen ASCII-Code (Siehe Anlage K) <u>gerade</u> ist:

```
1ØØ POKE  -16368, Ø : REM RESET KEYBOARD STROBE (HIGH BIT)
1Ø5 REM   PAUSE UNTIL KEYBOARD STROBE IS SET BY ANY KEY
11Ø WAIT  -16384, 128 : REM WAIT UNTIL HIGH BIT IS ONE
115 REM   PAUSE SOME MORE UNTIL KEY STRUCK IS EVEN
12Ø WAIT  -16384, 1, 1 : REM WAIT UNTIL LOW BIT IS ZERO
13Ø PRINT "EVEN"
14Ø GOTO 1ØØ
```

CALL imm & def

CALL aexpr

Diese Anweisung löst die Ausführung eines Maschinenunterprogramms auf der Speicherstelle aus, deren dezimale Adresse durch \aexpr\ festgelegt ist.

\aexpr\ muß sich im Bereich -65535 bis 65535 bewegen, ansonsten wird die Fehlermeldung
? ILLEGAL QUANTITY ERROR
angezeigt. In der Praxis wird \aexpr\ gewöhnlich durch den Bereich von Adressen begrenzt, für die gültige Speichergeräte vorhanden sind. Dieser Bereich erstreckt sich also von Ø bis zum maximalen Wert des HIMEM: Ihres Computers. Siehe dazu auch HIMEM: und POKE.

Äquivalente positive und negative Adressen können abwechselnd benutzt werden. So sind beispielsweise "CALL -936" und "CALL 646ØØ" identisch.
In der Anlage J finden sich Beispiele für den Gebrauch von CALL.

HIMEM: imm & def

HIMEM: aexpr

Stellt die Adresse der obersten Speicherstelle ein, die für ein BASIC-Programm zur Verfügung steht. Es wird benutzt, um den Raum des Speichers oberhalb des HIMEM: für Daten, graphische Darstellungen oder maschinensprachliche Unterprogramme zu schützen.

\aexpr\ muß sich im Bereich von -65535 bis einschließlich 65535 bewegen, damit die Anzeige der Fehlermeldung
? ILLEGAL QUANTITY ERROR
vermieden wird. Doch die Programme können nur dann zuverlässig ablaufen, wenn geeigne-

te Speicher-Hardware auf den Stellen vorhanden ist, die von den Adressen bis und einschließlich \aexpr\ spezifiziert werden.

Im allgemeinen wird der maximale Wert von aexpr von der Speicherkapazität des Computers bestimmt. Beispielsweise würde bei einem APPLE II mit 16 K-byte Speicherkapazität \aexpr\ 16384 betragen oder noch darunter liegen. Verfügt der APPLE II über 32 K-byte, dann könnte \aexpr\ bis zu 32768 gehen. Bei einem 48 K-byte-APPLE-II würde \aexpr\ bis 49152 betragen.

Normalerweise setzt das APPLESOFT automatisch HIMEM: auf die für den Nutzer oberste verfügbare Adresse, wenn APPLESOFT das erste Mal abgerufen wird.

Der laufende HIMEM:-Wert wird auf den Speicherstellen 116 und 115 (dezimal) gespeichert. Um den laufenden HIMEM:-Wert abzufragen, müssen Sie eingeben:
PRINT PEEK(116)* 256 + PEEK(115)

Wenn durch HIMEM: eine oberste Speicheradresse gesetzt wird, die unter der durch LOMEM: gesetzten Adresse liegt oder die für den Ablauf des Programms nicht genügend freien Speicherraum zuläßt, dann wird die Fehlermeldung:
? OUT OF MEMORY
angezeigt.

\aexpr\ kann sich im Bereich von Ø bis 65535 oder im äquivalenten Bereich -65535 bis -1 bewegen. Äquivalente positive und negative Werte können abwechselnd verwendet werden.

HIMEM: wird nicht durch CLEAR, RUN, NEW, DEL, durch das Verändern oder das Hinzufügen von Programmzeilen oder durch reset nullgestellt. HIMEM: wird nullgestellt durch reset ctrl B return, wodurch aber gleichzeitig jedes gespeicherte Programm gelöscht wird.

LOMEM: imm & def
LOMEM: aexpr

Stellt die Adresse der untersten Speicherstelle ein, die für ein BASIC-Programm zur Verfügung steht. Normalerweise ist das die Adresse der Speicheranfangsstelle für die erste BASIC-Variable.
APPLESOFT stellt normalerweise das LOMEM: auf das Ende des gerade laufenden Programms ein, bevor das Programm ausgeführt wird.

Dieser Befehl ermöglicht den Schutz von Variablen vor den Gefahren, die sich bei Computern mit großen Speichern im Zusammenhang mit den graphischen Darstellungen bei hoher Auflösung ergeben.

\aexpr\ muß sich im Bereich von -65535 bis einschließlich 65535 bewegen, andernfalls erscheint die Fehlermeldung
? ILLEGAL QUANTITY ERROR
auf dem Monitor. Wird das LOMEM: aber auf einen Wert eingestellt, der über dem des laufenden HIMEM: liegt, dann wird
? OUT OF MEMORY ERROR
angezeigt. Das bedeutet, daß \aexpr\ unter dem Maximalwert liegen muß, der für HIMEM: gesetzt werden kann. (Zur Erläuterung des Maximalwertes siehe HIMEM:)

Wenn LOMEM: unter die Adresse der obersten Speicherstelle eingestellt wird, die vom laufenden Operationssystem (zuzüglich irgendeines gespeicherten Programmes) belegt wird, dann wird wieder die Fehlermeldung
? OUT OF MEMORY ERROR
angezeigt. Das bringt eine absolute untere Grenze für \aexpr\ mit sich, die bei etwa 2Ø51 für APPLESOFT-Firmware liegt.

LOMEM: wird durch NEW, DEL, durch das Einfügen oder Verändern einer Programmzeile nullgestellt. LOMEM: wird desweiteren durch reset ctrl B nullgestellt, was aber gleichzeitig das Löschen aller gespeicherten Programme mit sich bringt. Es wird nicht nullgestellt durch RUN, reset ctrl C return oder reset ØG return.

Der laufende Wert des LOMEM: wird auf den Speicherstellen 1Ø6 und 1Ø5 (dezimal) gespeichert. Um den laufenden Wert von LOMEM: abzurufen, müssen Sie eingeben:
PRINT PEEK(1Ø6)*256 + PEEK(1Ø5)

Wenn es erst einmal gesetzt wurde, kann LOMEM: nur auf einen neuen Wert gesetzt werden, der oberhalb (im Speicher) des alten Wertes liegt, sofern es nicht erst noch durch eines der o.g. Kommandos nullgestellt wurde. Der Versuch, ein LOMEM: zu setzen, das unterhalb des noch gültigen liegt, hat die Fehleranzeige
? OUT OF MEMORY ERROR
zur Folge.

Die Veränderung des LOMEM: bei laufendem Programm kann zur Folge haben, daß bestimmte Stacks oder Teile vom Programm nicht verfügbar sind, so daß das Programm nicht ordentlich ablaufen könnte. Äquivalente positive und negative Adressen können abwechselnd verwendet werden.

USR imm & def

USR (aexpr)

Mit dieser Operation wird \aexpr\ an ein maschinensprachliches Unterprogramm übergeben.

Das Argument aexpr wird berechnet und in den Gleitkomma-Akkumulator (Stellen $9D bis $A3) gegeben. Dann wird ein JSR zur Stelle $ØA ausgeführt. Die Stellen $ØA bis $ØC müssen einen JMP zur Anfangsstelle des maschinensprachlichen Unterprogramms enthalten. Der sich aus der Operation ergebende Wert wird in den Gleitkomma-Akkumulator aufgenommen.

Um eine 2-byte-umfassende ganze Zahl aus dem Wert im Gleitkomma-Akkumulator zu erhalten, müßte Ihr Unterprogramm einen JSR nach $E1ØC machen. Auf return hin wird der ganzzahlige Wert in den Stellen $AØ (höherwertiges Byte) und $A1 (niedrigwertiges Byte) sein.

Um ein ganzzahliges Ergebnis in das Gleitkomma-Äquivalent umzuwandeln, so daß die Operation diesen Wert auswerfen kann, müssen Sie die 2-Byte-umfassende ganze Zahl in die Register A (höherwertiges Byte) und Y (niedrigwertiges Byte) einbringen. Lassen Sie jetzt einen JSR auf $E2F2 ausführen. Bei return wird der Wert in Gleitkommaschreibweise im Gleitkomma-Akkumulator sein.

Um zu APPLESOFT zurückzukehren, müssen Sie ein RTS ausführen.

Hier nun ein ganz einfaches Programm, bei dem die USR-Operation ausgeführt wird, nur um Ihnen die Form zu zeigen:

```
] reset
* ØA:4C ØØ Ø3 return
* Ø3ØØ:6Ø return
* ctrl C return
] PRINT USR(8)* 3
24
```

Auf der Stelle $0A lassen wir einen JMP (Code 4C) auf Stelle $300 (niedrigwertiges Byte vor höherwertigem Byte) ausführen.
Auf Stelle $300 legen wir einen RTS (Code 60). Zurück im APPLESOFT wurde das Argument 8 in den Akkumulator gegeben, wenn das USR(8) auftrat. Der Monitor führte einen JSR auf Stelle $0A aus, wo er auf einen JMP auf $300 traf. Auf $300 traf er auf ein RTS, das ihn zu APPLESOFT zurückschickte. Der ausgeworfene Wert war lediglich der ursprüngliche Wert 8 im Akkumulator, den APPLESOFT mit 3 multiplizierte, um schließlich 24 zu erhalten.

KAPITEL 4

Schreibbefehle und Befehle zum Aufbau

LIST imm & def

LIST [linenum1] [- linenum2]
LIST [linenum1] [, linenum2]

Wenn es weder linenum1 und linenum2 gibt, sei nun ein Trennsymbol vorhanden oder nicht, wird das gesamte Programm auf dem Schirm angezeigt.
Gibt es linenum1 ohne Trennsymbol oder gilt linenum1=linenum2, dann wird nur die Zeile mit linenum1 angezeigt.
Sind linenum1 und ein Trennsymbol vorhanden, dann wird das Programm beginnend mit Zeile linenum1 bis zu Ende hin angezeigt.
Sollte es nur ein Trennsymbol und linenum2 geben, dann wird das Programm vom Beginn an bis zu der Zeile linenum2 angezeigt.
Sollte es linenum1, ein Trennsymbol und linenum2 geben, dann wird das Programm von linenum1 bis einschließlich linenum2 aufgelistet.

Müssen mehr als eine Zeile aufgelistet werden, falls die mit linenum1 bezeichnete Zeile in der LIST-Anweisung nicht im Programm vorkommen sollte, wird durch die LIST-Anweisung die Zeile mit der nächsthöheren Zeilennummer, die im Programm vorkommt, aufgenommen.
Sollte die mit linenum2 im LIST-Befehl bezeichnete Zeile nicht im Programm vorkommen, wird vom LIST-Befehl die Zeile mit der nächsten darunterliegenden Zeilennummer, die im Programm vorkommt, aufgenommen.

Diese LIST-Anweisungen lassen das gesamte Programm anzeigen:

LIST Ø　　LIST [, I -] Ø　　LIST Ø [, I -] Ø

LIST linenum, Ø

listet das Programm von mit linenum bezeichneter Zeile bis zum Ende auf.
Durch die Anweisung
LIST , Q
wird das gesamte Programm aufgelistet, und dann erscheint die Fehlermeldung:

? SYNTAX ERROR .

Durch APPLESOFT werden die Programmzeilen mit Zeichen belegt, bevor sie gespeichert werden, wobei unnötiger Platz in diesem Prozeß eingespart wird. Beim Auflisten stellt APPLESOFT die mit Zeichen belegten Programmzeilen wieder her, wobei Leerräume entsprechend den eigenen Regeln eingefügt werden.

Beispielsweise wird dann

1Ø C=+5/-6:B=-5 zu 1Ø C = + 5 / - 6:B = - 5,

wenn aufgelistet wird.

Bei LIST kommen eine veränderbare Zeilenbreite und verschiedene Einschübe zur Anwendung. Das kann sich als problematisch erweisen, wenn versucht wird, eine aufgelistete Anweisung zu redigieren oder zu kopieren. Damit beim LIST der Aufbau mit zusätzlichen Freiräumen unterbunden wird, muß der Bildschirm gelöscht werden und das Textschreibfeld auf die 33-er Breite (das ist das Maximum) eingestellt sein. Das wird folgendermaßen gemacht:

HOME
POKE 33,33

ACHTUNG!

Durch APPLESOFT wird die Zeile auf 239 Zeichen verändert. Durch LIST werden dann Freiräume eingefügt. Auf diese Weise kann man viele zusätzliche Zeichen einfügen, wenn die Freiräume weggelassen werden. Bei der Eingabe von LIST werden sie erneut mit eingefügt. Der Versuch, die erweiterte Anweisung direkt vom Bildschirm zu kopieren, führt wieder zu der Veränderung auf 239 Zeichen einschließlich der Freiräume, die durch LIST eingefügt werden.

Das Auflisten wird durch ctrl C unterbrochen.

DEL imm & def
DEL linenum1 , linenum2

Durch DEL wird der ganze Zeilenbereich von linenum1 bis einschließlich linenum2 gelöscht. Sollte linenum1 nicht als Programmzeile existieren, dann wird die Zeile im Programm mit der nächsthöheren Zeilennummer an Stelle von linenum1 verwendet.

Gibt es linenum2 nicht im Programm, dann wird die nächstniedrigere Zeilennummer benutzt.

Wenn man nicht der gewöhnlichen Struktur treu bleibt, dann werden durch DEL die verschiedenen nachstehend aufgeführten Folgen eintreten:

Syntax	Ergebnis
DEL	? SYNTAX ERROR
DEL,	? SYNTAX ERROR
DEL , b	? SYNTAX ERROR
DEL -a[, b]	? SYNTAX ERROR
DEL Ø,b	löscht Zeile Ø unabhängig vom Wert von b.
DEL 1,-b	wird außer Acht gelassen, selbst wenn die unterste Programmzeilennummer die Ø ist.
DEL a,-b	? SYNTAX ERROR, wenn a größer ist als die unterste Programmzeilennummer, wenn nicht die unterste Programmzeilennummer die Ø ist und a=1 ist.
DEL a,-b	wird nicht berücksichtigt, wenn a ungleich Ø und die einzige Programmzeile die Zeile Ø ist

ACHTUNG!

DEL a,-b	wird nicht berücksichtigt, wenn a ungleich Ø und a kleiner als oder gleich der untersten Programmzeilennummer ist.

ACHTUNG!

DEL a[,]	wird nicht berücksichtigt.

ACHTUNG!

DEL a,b	wird nicht berücksichtigt, wenn a ungleich Ø und größer als b ist.

ACHTUNG!

Wenn DEL in verzögerter Ausführungsweise benutzt wird, dann arbeitet es wie oben beschrieben, es unterbricht die Programmausführung. Das CONT wird in dieser Situation

nichts nützen.

REM imm & def

REM {Zeichen | "}

Damit wird das Einfügen irgendeines Textes in das Programm ermöglicht. Alle Zeichen, einschließlich der Anweisungstrennsymbole und Leerzeichen, können dabei vorkommen. Ihre sonst übliche Bedeutung wird nicht berücksichtigt. Ein REM wird nur durch das return beendet.

Wenn die REM aufgelistet werden, dann wird durch APPLESOFT ein zusätzlicher Freiraum nach dem REM eingefügt, unabhängig davon, ob und wieviel Freiräume vom Benutzer nach dem REM eingegeben wurden.

VTAB imm & def

VTAB aexpr

Damit wird der Cursor zu der Zeile bewegt, die so viele Zeilen unterhalb der obersten Zeile auf dem Bildschirm liegt, wie \aexpr\angibt. Die oberste Zeile ist Zeile 1, die unterste Zeile ist Nr. 24.
Diese Anweisung kann auch die Bewegungsrichtung des Cursors, entweder hinauf oder herunter, enthalten, aber niemals die seitliche Bewegungsrichtung (links oder rechts).
Argumente außerhalb des Bereiches 1 bis 24 ergeben die Fehlermeldung:

? ILLEGAL QUANTITY ERROR

Bei VTAB werden absolute Bewegungen ausgeführt, die sich nur auf Ober- und Unterkante des Bildschirms beziehen. Das Textfeld wird nicht berücksichtigt. Im Graphik-Betrieb wird VTAB den Cursor auf dem für graphische Darstellungen vorgesehenen Feld des Bildschirms bewegen. Wenn durch VTAB der Cursor auf die Zeile unterhalb des Textfeldes bewegt wird, dann wird alles, was danach auszudrucken ist, auf dieser Zeile ausgedruckt.

HTAB imm & def

HTAB aexpr

Nehmen wir an, die Zeile, in der sich der Cursor befindet, hat 255 Positionen, nämlich 1 bis 255. Unabhängig von der Textfeldbreite, können Sie gesetzt haben, daß sich die Positionen 1 bis 40 auf der laufenden Zeile, 41 bis 80 auf der nächsten unterhalb liegenden Zeile usw. befinden. Durch HTAB wird der Cursor auf die Position bewegt, die sich \ aexpr\ Positionen vom linken Rand der Bildschirmzeile entfernt befindet. Die HTAB-Bewegungen beziehen sich immer auf die linke Randbegrenzung des Textfeldes, sind aber unabhängig von der Zeilenbreite. HTAB kann auch den Cursor außerhalb des Textfeldes bewegen, aber nur so lange, wie das Drucken eines Zeichens dauert. Um den Cursor in die linksbündige Position der laufenden Zeile zu bringen, benutzen Sie:

HTAB 1.

ACHTUNG!

HTAB Ø bewegt den Cursor auf die Position 256!

Wenn \ aexpr \ negativ oder größer als 255 ist, so erscheint die Fehlermeldung:

? ILLEGAL QUANTITY ERROR

Es ist zu beachten, daß die Strukturen von HTAB und VTAB nicht gleichlaufend zu behandeln sind, insofern als nämlich die HTAB über die rechte Randbegrenzung des Bildschims hinaus nicht die Fehlermeldung

? ILLEGAL QUANTITY ERROR

verursachen, sondern den Cursor auf die nächste, darunterliegende Zeile springen lassen und tabellieren

((aexpr-1)MOD 4Ø)+1

TAB imm & def

TAB (aexpr)

TAB muß in einer PRINT-Anweisung benutzt werden, und aexpr muß in Klammern gesetzt werden. Durch TAB wird der Cursor auf die Position bewegt, die\aexpr\ Druckpositionen

von der linken Randbegrenzung des Textfeldes entfernt ist, sofern \aexpr\ größer ist als der Wert der gegenwärtigen Cursorposition in Bezug auf den linken Rand. Wenn \aexpr\ kleiner als der Wert der gegenwärtigen Cursorposition ist, dann wird der Cursor nicht bewegt - TAB bewegt den Cursor niemals nach links (man benutze dazu HTAB).

Wenn durch TAB der Cursor über die rechte Grenze des Textfeldes hinaus bewegt wird, stellt sich der Cursor auf die linksbündige Position der nächsten Zeile des Textfeldes ein, und der Zeilentransport beginnt von dort.

ACHTUNG!

TAB(Ø) bewegt den Cursor auf Position 256!

\ aexpr\ muß im Bereich von Ø bis 255 liegen. Andernfalls wird die Fehlermeldung

? ILLEGAL QUANTITY ERROR

angezeigt.

TAB wird als Tabuwort bestimmt, aber nur wenn das nächste Zeichen, das sich ohne Zwischenraum anschließt, die nach rechts geöffnete Klammer ist.

POS imm & def

POS (expr)

Hiermit wird die gegenwärtige Position des Cursors in der Horizontalen des Bildschirms angegeben. Die Position bezieht sich auf den linken Rand des Textfeldes. Beim linken Rand wird die Ø wiedergegeben. Obwohl expr lediglich vorhanden ist, um einen Ausdruck in der Klammer zu haben, der die beiden Klammern auseinanderhält, wird er dennoch bearbeitet. Deshalb darf es kein unzulässiger Ausdruck sein. Alles, was sich als eine Zahl, eine Gruppe oder als Variablenname auslegen läßt, kann für expr verwendet werden. Wenn expr aus Zeichen gesetzt ist, die nicht als ein Variablenname angesehen werden können, dann müssen diese Zeichen in Anführungsstriche gesetzt werden.
Beachten Sie, daß für die HTAB- und TAB-Anweisungen die Positionen mit 1 beginnend gezählt werden, aber für POS und SPC mit Ø beginnend numeriert sind.

Deshalb wird durch

PRINT TAB(23); POS(Ø)

22 gedruckt, wohingegen durch

PRINT SPC(23); POS(Ø)

23 gedruckt wird.

SPC imm & def

SPC (aexpr)

SPC muß in einer PRINT-Anweisung verwendet werden, und aexpr muß eingeklammert sein. Es werden dadurch\aexpr\ viele Abstände zwischen dem vorher Gedruckten (oder bei Versäumnis zwischen dem linken Rand des Textfeldes) und dem als nächstes zu Druckendem eingefügt, wenn der SPC-Befehl mit den vorausgehenden und nachfolgenden Posten durch Nebeneinanderstellen oder durch Semikolons miteinander verkettet ist. Durch SPC(Ø) wird kein Abstand eingefügt.

\aexpr\ muß im Bereich von Ø bis einschließlich 255 liegen. Andernfalls erscheint auf der Anzeige die Fehlermeldung:

? ILLEGAL QUANTITY ERROR

Allerdings können auch mehrere SPC miteinander verkettet werden, was dann so aussieht:

PRINT SPC(25Ø)SPC(139)SPC(255) usw.,

um einmal willkürlich große positive Abstände zu zeigen.
Zu beachten ist, daß beim SPC (aexpr) der Cursor eine gegebene Anzahl von Abständen zwischen dem vorher gedrucktem Posten und dem nächsten vorgerückt wird, während bei HTAB der Cursor auf eine absolute Position des Bildschirms bewegt wird, die sich auf den linken Rand des Textfeldes bezieht. Daraus ergibt sich, daß bei SPC die neue Position sich irgendwo im Textfeld befinden kann und nur von dem vorher gedruckten Posten abhängig ist.

Der Zeilenvorschub über die rechte Seitenbegrenzung des Textfeldes hinaus führt dazu, daß der Zeilenvorschub oder das Drucken von Zeichen auf der nächsten Zeile des Textfeldes

fortgesetzt wird.

Beim Ausfüllen von Tabellen kann der Zeilenvorschub innerhalb einer Tabellenspalte oder von einer in die andere Tabellenspalte oder gar über eine ganze Tabellenspalte hinweg erfolgen.

Wenn\aexpr\ eine reelle Zahl ist, dann wird sie in eine ganze Zahl umgewandelt.

SPC wird als Tabuwort nur dann identifiziert, wenn das ohne Zwischenraum folgende Zeichen die nach rechts geöffnete Klammer ist.

HOME imm & def

HOME

Ohne folgende Parameter. Der Cursor wird in die obere linksbündige Stellung des Schreibfeldes bewegt, und der gesamte Text innerhalb des Feldes wird gelöscht. Dieser Befehl ist mit dem "CALL-936" und "esc@ return" identisch.

CLEAR imm & def

CLEAR

Ohne folgende Parameter. Stellt alle Variablen und Bereiche sowie Gruppen null. Bringt die Zeiger und Stacks in die Ausgangsstellung.

FRE imm & def

FRE (expr)

FRE gibt die Speicherkapazität (in Byte) wieder, die dem Nutzer noch zur Verfügung steht. Man kann manchmal zum Schluß noch mehr Speicherkapazität über haben, als erwartet wird, denn APPLESOFT speichert kopierte Gruppen nur einmal, d.h. daß, wenn A$="PIPPIN" und B$="PIPPIN", die Gruppe "PIPPIN" nur einmal gespeichert wird.

Sollte die Anzahl der freien Bytes im Speicher über 32767 hinausgehen, dann gibt FRE (expr) eine negative Zahl wieder. Wenn man zu dieser Zahl 65536 addiert, dann kennt man die genaue Anzahl der freien Bytes im Speicher.

FRE (expr) gibt die Anzahl der freien Bytes an, die noch unterhalb des Gruppenspeicherraums und oberhalb des Bereichsraums für Zahlenbereiche und Gruppenzeiger verbleiben (siehe auch Speicherplan in Anlage 1). HIMEM: kann bis auf 65535 gesetzt werden, doch wenn es über die oberste RAM-Speicherstelle in Ihrem APPLE-Computer gesetzt wird, dann kann auf FRE hin eine ziemlich bedeutungslose Zahl wiedergegeben werden, die die Speicherkapazität überschreitet (siehe auch HIMEM: und POKE zwecks Erläuterung der Speichergrenzen).
Wenn der Inhalt einer Gruppe während des Programmlaufs verändert wird (d.h. A$="cat" wird zu A$="dog"), dann wird durch APPLESOFT "cat" nicht eliminiert, sondern einfach eine neue Datei für "dog" eröffnet. Folglich füllen eine ganze Reihe alter Zeichen langsam den Speicherraum vom HIMEM: abwärts bis zu oberen Grenze des Bereichsraums. APPLESOFT wird automatisch eine "Großreinigung" des Speichers vornehmen, wenn diese alten Eingaben in den freien Bereichsraum fließen sollten, aber wenn man irgendeinen Freiraum für Maschinenprogramme oder den Seitenpuffer für die Auflösung benutzt, dann könnten sie "fertiggemacht" werden.
Wenn die Anweisung

X = FRE (Ø)

in periodischen Abständen in Ihrem Programm verwendet wird, dann wird die "Großreinigung" erzwungen, und die oben beschriebenen Vorkommnisse treten nicht ein.
Obwohl expr nur einen beliebigen Klammerausdruck darstellt, der die Klammern auseinanderhält, wird es bearbeitet, und sollte demzufolge keine unzulässige Größe sein.

FLASH imm & def

INVERSE imm & def

NORMAL imm & def

FLASH

INVERSE

NORMAL

Die drei Kommandos werden benutzt zur Einstellung des Video-Output-Betriebs. Bei ihnen gibt es keine Parameter. Sie beeinflussen auch nicht die Anzeige von Zeichen,wenn Sie sie in den Computer eingeben, oder die Anzeige von Zeichen, die sich bereits auf dem Bildschirm befinden.

FLASH stellt den Video-Betrieb auf "Aufleuchten" um, so daß die Ausgabe des Computers abwechselnd weiß auf schwarzem Grund und umgekehrt auf dem Bildschirm erscheint.

INVERSE stellt den Video-Betrieb auf die Umkehrung der Ausgabe ein, so daß schwarze Zeichen auf weißem Grund angezeigt werden.

NORMAL stellt den Betrieb auf die gewöhnliche Art zurück, nämlich weiße Buchstaben werden auf schwarzem Grund sowohl für die Ein- als auch für die Ausgabe gedruckt.

SPEED imm & def

SPEED = aexpr

Hiermit wird die Geschwindigkeit eingestellt, mit der die Zeichen zum Bildschirm oder zu anderen Ein- und Ausgabegeräten übermittelt werden sollen. Die niedrigste Geschwindigkeit liegt bei Ø, die höchste bei 255. Werte außerhalb des Bereiches haben die Fehlermeldung

? ILLEGAL QUANTITY ERROR

zur Folge.

esc A	imm	& def	(nur beim Schreiben)
esc B	imm	& def	(nur beim Schreiben)
esc C	imm	& def	(nur beim Schreiben)
esc D	imm	& def	(nur beim Schreiben)

Die escape-Taste, die mit "ESC" beschriftet ist, kann in Verbindung mit der Buchstabentaste A, B, C oder D benutzt werden. um den Cursor zu bewegen. Um den Cursor um einen Abstand zu verschieben, drückt man zuerst die escape-Taste und dann wieder loslassen. Danach drückt man die entsprechende Buchstabentaste.

Kommando	Bewegung des Cursors um einen Abstand nach
esc A	rechts
esc B	links
esc C	unten
esc D	oben

Diese escape-Befehle bewirken nicht, daß die Zeichen mit dem Cursor verschoben werden; sie bleiben sowohl auf dem Bildschirm als auch im Speicher. Die escape-Befehle allein bewirken ebenfalls nicht, daß die Programmzeile gedruckt wird.

Um die Programmzeile zu verändern, muß man sie sich auflisten lassen. Benutzt man das escape-Kommando, um den Cursor genau auf das allererste Zeichen der aufgelisteten Zeile zu bringen. Dann betätigt man die Rechtspfeil- und REPT-Tasten, um die Zeichen vom Bildschirm abschreiben zu lassen. Man gebe immer, wenn der Cursor auf dem Zeichen steht, das verändert werden soll, ein anderes ein. Wenn die Zeile nicht aufgelistet wurde, dann wird das Ladesymbol ( ]), welches zu Anfang der Zeile erscheint, nicht kopiert. Abschließend betätigt man die RETURN-Taste, um die Zeile speichern oder ausführen zu lassen.

repeat nur imm (nur beim Schreiben)

Die repeat-Taste ist mit "REPT" beschriftet. Wenn man diese Taste gleichzeitig mit einer Zeichentaste drückt, dann wird das Zeichen wiederholt. Wenn man das erste Mal diese Taste allein drückt, dann wird das zuletzt gedruckte Zeichen wiederholt.

right arrow nur imm (nur beim Schreiben)
left arrow nur imm (nur beim Schreiben)

Die Rechtspfeiltaste bewegt den Cursor nach rechts. Beim Überstreichen der Zeichen, wird jedes einzelne auf dem Bildschirm im APPLE-II-Speicher kopiert, genau als würden Sie das Zeichen eingeben. Die Taste wird gemeinsam mit der repeat-Taste verwendet, um die Neueingabe einer ganzen Zeile mit nur geringfügigen Veränderungen zu ersparen.

Die Linkspfeiltaste bewegt den Cursor nach links. Jedesmal wenn der Cursor um eine Stelle nach links verschoben wird, wird ein Zeichen aus der Programmzeile gestrichen, die Sie gerade schreiben, unabhängig davon, worüber der Cursor streicht. Der Bildschirm wird dabei nicht berücksichtigt, und es ändert sich nichts in der Anzeige auf dem Bildschirm.

ACHTUNG!

Wenn Sie nicht gerade eine Zeile eingeben, für die noch kein return gedrückt wurde, kann mit der Linkspfeiltaste kein Zeichen einer laufenden Programmzeile gelöscht werden. In diesem Falle wird durch Betätigen der Taste das Ladesymbol (]) in Spalte Ø der nächsten Zeile, gefolgt vom Cursor erscheinen. Deshalb kann häufig der Cursor nicht in die Spalte Ø des Bildschirms gebracht werden, indem man die Linkspfeiltaste benutzt, denn es muß ein Zeichen einer laufenden Programmzeile mit jeder Bewegung gelöscht werden. Siehe escape-Kommandos zur Thematik: reine Verschiebungen ohne Löschen oder Kopieren.

ctrl X nur imm

Hiermit wird dem APPLE II angewiesen, die gegenwärtig eingegebene Zeile außer Acht zu lassen, ohne dabei irgendeine vorausgehende Zeile mit der gleichen Zeilennummer zu löschen. Ein Schrägstrich (\ ) wird am Ende angezeigt, um das Nichtberücksichtigen der Zeile anzukündigen. Der Cursor springt dann auf Spalte Ø der nächsten Zeile vor. Das Kommando kann auch während einer Anwort auf eine INPUT-Anweisung benutzt werden.

KAPITEL 5

Bereiche und Gruppen

DIM imm & def

DIM var Index [{,var Index}]

Wenn eine DIM-Anweisung ausgeführt wird, dann wird Platz für den Bereich mit den Namen var freigehalten. Zwei Bitgruppen im Speicher werden für die Speicherung eines Bereichsvariablennamens, zwei für die Größe des Bereichs, eine für die Anzahl der Dimensionen und zwei für jeweils eine Dimension verwendet. Wie unten erklärt, hängt derUmfang des Platzes, der für die Elemente des Bereichs reserviert wird, von der Art des Bereichs ab.

Indizies reichen von Ø bis \Index\ . Die Anzahl der Elemente in einen n-dimensionalen Bereich beträgt

(\Index1\ + 1) * (\Index2\ + 1)* ... * (\Index n\ + 1).

Beispielsweise werden durch DIM SHOW (4,5,3) 5 * 6 * 4 * Elemente = (120 Elemente) reserviert. Typische Elemente sind:

SHOW (4,4,1)
SHOW (Ø,Ø,2)
usw.

Die maximale Zahl von Dimensionen für einen Bereich beträgt 88, selbst wenn jede Dimension nur ein Element enthält: DIM A (Ø,Ø, ... Ø), in dem 89 mal die Ø steht, führt zur Fehlermeldung: ? OUT OF MEMORY ERROR, aber DIM A(Ø,Ø, ...Ø), in dem 88 mal die Ø steht, ist zulässig.

In der Praxis aber wird die Größe der Bereiche häufiger durch die zur Verfügung stehende Speicherkapazität begrenzt. Jedes ganzzahlige Bereichselement belegt 2 Byte (16 Bit) im Speicher. Jedes reellzahlige Bereichselement belegt 5 Byte (4Ø Bit). Gruppenvariable verbrauchen 3 Byte pro Element (1 Byte für die Länge, 2 Byte für den Stellenzeiger), der als ein ganzzahliger Bereich gespeichert wird, wenn der Bereich geDIMt wird. Da die Gruppen selbst durch das Programm gespeichert werden, benötigen sie zusätzlich noch 1 Byte pro Zeichen. Siehe auch unter Anhang .

Wird ein Bereichselement in einem Programm benutzt, noch ehe diese Variable geDIMt wurde,

dann wird durch APPLESOFT ein maximaler Index von 1Ø für jede Dimension in dem Index des Elements zugeordnet.

Wird eine Variable benutzt, deren Index größer ist als das gekennzeichnete Maximum oder die eine unterschiedliche Anzahl von Dimensionen verlangt, die nicht in einem DIM-Befehl festgelegt wurde, dann erscheint die Fehlermeldung:

?BAD SUBSCRIPT ERROR .

Wenn im Programm ein Bereich geDIMt wird, der den gleichen Namen wie ein schon vorher geDIMter Bereich hat (auch wenn fehlerhaft), dann erscheint die Fehlermeldung:

?REDIM'D ARRAY ERROR .

Die einzelnen Gruppen in einem Gruppenbereich werden nicht dimensioniert, aber können größer oder kleiner sein - je nach Notwendigkeit. Die Anweisung WARD$(5) = "ABCDE" schafft eine Gruppe mit der Länge 5. Die Anweisung WARD$(5) = ''' macht den Platz der derGruppe WARD$(5) reserviert wurde, wieder frei. Eine Gruppe kann bis maximal 255 Zeichen umfassen.

Die Bereichselemente werden durch die Ausführung von RUN oder CLEAR auf Null gesetzt.

LEN imm & def

LEN (sexpr)

Diese Operation gibt die Anzahl der Zeichen in einer Gruppe wieder, die zwischen Ø und 255 liegen kann. Wenn das Argument eine Verkettung von Gruppen ist, deren Länge über 255 hinausgeht, dann wird die Fehlermeldung ?STRING TOO LONG ERROR angezeigt.

STR$ imm & def

STR$ (aexpr)

Diese Funktion verwandelt \aexpr\ in eine Gruppe, die diesen Wert darstellt. aepr wird beechnet, ehe es in eine Gruppe umgewandelt wird. STR$ (1ØØ ØØØ ØØØ ØØØ) ergibt 1E+11. Wenn \aexpr\ die Grenzen für reelle Zahlen überschreitet, dann wird die Fehlermeldung ? OVERFLOW ERROR angezeigt.

VAL imm & def
VAL (sexpr)

Bei dieser Funktion wird versucht, eine Gruppe als eine reelle oder ganze Zahl zu interpretieren, wobei dann der Wert derZahl wiedergegeben wird. Das erste Zeichen der Gruppe muß ein möglicher Posten in einer Zahl sein (einleitende Freiräume sind erlaubt), sonst wird Ø wiedergegeben. Jedes Zeichen danach wird gleichermaßen untersucht, bis das erste definitiv nichtnumerische Zeichen vorkommt (Zwischenleerräume, Kommas, + und - sowie E sind alles mögliche numerische Zeichen im richtigen Kontext). Das erste nichtnumerische Zeichen und alle danach folgenden Zeichen werden nicht berücksichtigt, und die Gruppe wird bis zu diesem Punkt als eine reelle oder ganze Zahl berechnet.

Sollte eine Gruppenverkettung aus mehr als 255 Zeichen bestehen und das Argument von VAL sein, dann erscheint die Fehlermeldung: ? STRING TOO LONG ERROR .
Wenn ein absoluter Wert einer Zahl, die wiedergegeben wird, größer als 1E38 sein sollte, oder wenn die Zahl mehr als 38 Stellen hat (einschließlich der aufeinanderfolgenden Nullen), dann wird ?OVERFLOW ERROR angezeigt.

CHRS imm & def
CHRØ (aexpr)

Das ist eine Funktion, die das ASCII-Zeichen wiedergibt, das dem Wert von aexpr entspricht. \aexpr\ muß im Bereich von Ø bis einschließlich 255 liegen. Anderenfalls kommt es zur Fehlermeldung: ? ILLEGAL QUANTITY ERROR
Reelle werden in ganze Zahlen umgewandelt.

ASC imm & def
ASC (sexpr)

Diese Funktion gibt einen ASCII-Code (nicht notwendigerweise die niedrigste Zahl) für das erste Zeichen von \sexpr\ wieder. Die ASCII-Codes von 96 bis 255 erzeugen auf dem APPLE Zeichen, die die von Ø bis 95 wiederhol en. Obwohl jedoch CHRØ(65) ein A wiedergibt, ebenso wie CHRØ(193), werden diese beiden A von APPLESOFT nicht als das gleiche Zeichen identifiziert, wenn logische Gruppenoperatoren benutzt werden. Wenn eine Gruppe das Argument ist, dann muß sie in Anführungszeichen gesetzt werden. Die Anführungszei-

chen brauchen nicht in der Gruppe enthalten zu sein. Ist die Gruppe Null, dann erscheint die Fehlermeldung: ? ILLEGAL QUANTITY ERROR .

ACHTUNG!

Der Versuch, die ASC-Funktion auf ctrl@ zu benutzen, ergibt ? SYNTAX ERROR .

LEFT$ imm & def
LEFT$ (sexpr, aexpr)

Diese Funktion gibt die ersten (linksbündigen) \aexpr\ Zeichen von \sexpr\ wieder.

PRINT LEFT$("APPLESOFT", 5)
APPLE

Kein Teil dieses Befehls kann ausgelassen werden. Wenn \aexpr\ < 1 oder \aexpr\ > 255, dann kommt es zur Fehlermeldung: ? ILLEGAL QUANTITY ERROR

Ist \aexpr\ eine reelle Zahl, wird sie in eine ganze umgewandelt.
Falls \aexpr\ > LEN(sexpr), dann werden nur die Zeichen, die die Gruppe bilden, wiedergegeben. Alle anderen zusätzlichen Positionen werden nicht berücksichtigt.

Sollte "$" aus dem Befehl ausgelassen werden, dann wird APPLESOFT LEFT wie einen arithmethischen Variablennamen behandeln,und die Meldung ? TYPE MISMATCH ERROR wird angezeigt.

RIGHT$ imm & def
RIGHT$ (sexpr, aexpr)

Diese Funktion gibt die letzten (am weitesten rechts stehenden) \aexpr\ Zeichen von \sexpr\ wieder.

PRINT RINGHT$ ("APPLESOFT" + "WARE", 8)
SOFTWARE

Nicht ein Teil dieses Befehls kann ausgelassen werden. Wenn \aexpr\ >= LEN (sexpr),

dann gibt RIGHT$ die gesamte Gruppe wieder. Die Meldung ? ILLEGAL QUANTITY ERROR wird angezeigt, wenn \aexpr\ < 1 oder \aexpr\ > 255.

RIGHT$ (sexpr, aexpr) = MID$(sexpr, LEN(sexpr)+1 - \aexpr\)

Wenn "$" aus dem Befehl ausgelassen wird, dann behandelt APPLESOFT RIGHT wie einen Namen einer arithmetischen Variablen, und die Meldung ? TYPE MISMATCH ERROR wird angezeigt.

MID$ imm & def
MID$ (sexpr, aexpr1 [, aexpr 2])

MID$ mit zwei Argumenten gibt die Untergruppe, beginnend bei dem \aexpr1\-ten Zeichen von \sexpr\ bis zum letzten Zeichen von \sexpr\ wieder.

PRINT MID$("APPLESOFT",3)
PLESOFT
MID$(sexpr, aexpr) = RIGHT$(sexpr, LEN(sexpr)+1 - \aexpr\)

MID$ mit drei Argumenten gibt \aexpr2\ Zeichen von \sexpr\, beginnend bei dem \aexpr1\-ten Zeichen und nach rechts fortschreitend, wieder.

PRINT MID$("APPLESOFT",3,5)
PLESO

Wenn \aexpr1\ > LEN(sexpr), dann gibt MID$ eine Nullgruppe wieder. Wenn \aexpr1\ + \aexpr2\ über die Länge von \sexpr\ (oder 255, der Maximallänge einer Gruppe) hinausgeht, wird alles Zusätzliche nicht berücksichtigt. MID$(A$,255,255) gibt ein Zeichen wieder, wenn LEN(A$) = 255. Anderenfalls wird die Nullgruppe wiedergegeben.

Wenn entweder \aexpr1\ oder \aexpr2\ außerhalb des Intervalls Ø bis einschließlich 255 liegt, dann wird die Fehlermeldung ? ILLEGAL QUANTITY ERROR angezeigt. Wenn "$" aus dem Befehl ausgelassen wurde, dann wird das MID durch APPLESOFT wie ein Name einer arithmetischen Variablen behandelt, und die Fehlermeldung ? TYPE MISMATCH ERROR erscheint auf der Anzeige.

STORE imm & def
RECALL imm & def

STORE avar
RECALL avar

Mit dem Befehl "STORE" wird ein Bereich auf die Magnetbandkassette aufgezeichnet, mit "RECALL" wird er wieder abgerufen. Die Namen der Bereiche werden nicht mit ihren Werten gespeichert, so daß ein Bereich abgerufen werden kann, wobei ein anderer Name verwendet wird als in dem STORE-Befehl.

Die Dimensionen der Bereiche, die vom RECALL-Befehl benannt werden, sollten mit den Dimensionen identisch sein, mit denen der ursprüngliche Bereich geSTOREt wurde. Wenn beispielsweise ein durch DIM A(5,5,5) dimensionierter Bereich gespeichert wird, dann kann man ihn in einen Bereich, der durch DIM B (5,5,5) dimensioniert ist, abrufen. Wenn das nicht beachtet wird, dann wird es zu einem Zahlendurcheinander im abgerufenen Bereich, zu zusätzlichen Nullen in dem Bereich oder zur Fehlermeldung "? OUT OF MEMORY ERROR" kommen.

Im allgemeinen wird die Fehlermeldung "? OUT OF MEMORY ERROR" nur dann angezeigt, wenn die Gesamtanzahl der für den abberufenen Bereich res ervierten Elemente nicht ausreicht, um alle Elemente des Bereichs, die im STORE-Befehl angesprochen wurden, darin unterzubringen.

DIM A (5,5,5)
SOTRE A

reserviert für 6*6*6 Elemente Platz auf dem Kassententonband.

DIM B (5,35)
RECALL B

wird zu einer Fehlermeldung "ERR" führen, und ein Zahlendurcheinander im Bereich B entsteht. Das Programm wird aber weiterlaufen.

D ieAnweisung aber

DIM B (5,25)
RECALL B

würde die Fehlermeldung ? OUT OF MEMORY ERROR zur Folge haben, wonach die Ausführung des Programms eingestellt wird. In diesem Falle enthielt der Bereich B 6* 26 Elemente, also zu wenig, um alle Elemente des Bereiches A umfassen zu können.

Wenn der abberufene Bereich über eine gleiche Anzahl von Dimensionen verfügt [DIM A (5,5,5) definiert einen Bereich mit drei Dimensionen mit der jeweiligen Größe 6] wie der gespeicherte Bereich, kann jede Dimension des abberufenen Bereichs größer sein als die entsprechende Dimension des gespeicherten Bereichs. Ein Durcheinander von Zahlen wird jedoch dann im abberufenen Bereich auftreten, wenn es nicht die letzte Dimension des abberufenen Bereichs ist, die größer ist als die letzte Dimension des gespeicherten Bereichs. Auf jeden Fall werden Sie zusätzliche Nullen finden, die in den überzähligen Elementen des abberufenen Bereichs gespeichert sind, aber nur in diesem letzten Fall werden Sie die Nullen an jenen Stellen finden, wo Sie sie erwartet haben. Nachdem ein Bereich gespeichert wurde mit

DIM A (5,5,5)
STORE A,

werden Sie erkennen, daß sowohl

DIM B (1Ø,5,5)
RECALL B

als auch

DIM B (5,1Ø,5)
RECALL B

den Bereich B mit durcheinandergebrachten Zahlen aus dem Bereich A füllen, während

DIM B (5,5,1Ø)
RECALL B

ordentlich funktioniert, wobei sich für die zusätzlichen Elemente im Bereich B Nullen einstellen.

Wir erörtern also zwei "Regeln" für das Speichern (STORE) und das Abberufen (RECALL) von Bereichen mit gleicher Dimensionsanzahl, nämlich :

1. Nur die letzte Dimension des abberufenen Bereichs kann größer als die letzte Dimension im gespeicherten Bereich sein.

2. Die Gesamtanzahl der abberufenen Elemente muß mindestens der Anzahl der gespeicherten Elemente gleich sein.

Wenn man Regel 1 hinsichtlich der Dimensionen, die für beide Bereiche gleich sind (das muß jeweils die erste Dimension sein), und Regel 2 befolgt, dann kann man einen Bereich mit mehr Dimensionen abberufen, als man gespeichert hat. Eine ERR-Meldung wird angezeigt, aber die Programmausführung läuft weiter.

DIM B (5,5,5,5)
RECALL B
wird einwandfrei im o. g. Beispiel funktionieren (nach der ERR-Meldung, und mit vielen zusätzlichen Nullen im Bereich B), aber

DIM B (5,5,3,5)
RECALL B
wird den Bereich B mit einem Zahlendurcheinander füllen (nachdem die ERR-Meldung gekommen war), und

DIM B (5,5,1,1)
RECALL B
wird die Fehlermeldung ?OUT OF MEMORY ERROR zur Folge haben, denn 6*6*2*2 Elemente im Bereich B sind weniger als 6*6*6 Elemente, die im Bereich A gespeichert wurden.

Es können nur reell- und ganzzahlige Bereiche gespeichert werden. Gruppenbereiche müssen unter Verwendung der ASC-Funktion in einen ganzzahligen Bereich umgewandelt werden, damit man sie speichern kann.

Obwohl sich STORE und RECALL auf ihre Variablen ohne Angabe von Indizies oder Dimensionen beziehen, können nur Bereiche mit Hilfe von STORE gespeichert und mit Hilfe

von RECALL abberufen werden. Das Programm

```
1ØØ A (3) = 45
11Ø A = 27
12Ø STORE A
```

wird die Bereichselemente A(Ø) bis A(1Ø) auf Band aufzeichnen, denn bei Verabsäumen von DIM wird der Bereich auf 11 Elemente dimensioniert, und nicht die Variable A (die in dem Programm 27 heißt).

Durch STORE-Anweisungen wird weder eine Lademeldung noch irgendein anderes Signal ausgelöst. Der Benutzer muß dafür sorgen, daß der Recorder auf Aufnahme eingestellt ist, wenn die Anweisung ausgeführt wird. Der "Piep"-Ton signalisiert den Anfang und das Ende der Aufnahme.

Das Programm

```
3ØØ DIM B (5,13)
31Ø B = 4
32Ø RECALL B
```

liest vom Band die (6*14) 84 Bereichselemente B(Ø,Ø) bis B(5,13). Der Wert der Variablen B wird nicht verändert. Es gibt auch hier keine Lademeldung. Der"Piep"-Ton signalisiert Anfang und Ende der Wiedergabe.

Wenn entweder STORE oder RECALL einen Bereichsnamen enthalten, der nicht vorher DIMensioniert oder mit einem Index verwendet wurde, dann wird die Fehlermeldung ? OUT OF DATA ERROR angezeigt. Bei der Sofortausführung muß, wenn sich entweder STORE oder RECALL auf einen Bereichsnamen beziehen, der in einer Programmzeile definiert wird, die im verzögerten Ausführungsbetrieb läuft, erst einmal die Zeile in der verzögerten Ausführung bearbeitet werden, bevor STORE oder RECALL eingegeben werden können.

Nur reset kann das STORE und das RECALL unterbrechen. Wenn die Tabuwörter STORE und RECALL als die ersten Zeichen irgendeines Variablennamens benutzt werden, können die Befehle ausgeführt werden, noch ehe irgendeine Fehlermeldung in der Form ? SYNTAX ERROR angezeigt wird. Die Anweisung STOREHOU SE=5 wird die Fehlermeldung ? OUT OF DATA ERROR mit sich bringen, es sei denn, es wurde vorher ein Bereich definiert, dessen Name mit den Zeichen HO beginnt.

Im letzteren Fall wird APPLESOFT versuchen, den Bereich speichern zu lassen. Sie werden dann zunächst den "Piep"-Ton ein erstes und ein zweites Mal vernehmen und schließlich die Fehlermeldung ?SYNTAX ERROR auf der Anzeige erkennen, da APPLESOFT versucht, den Rest der Anweisung, nämlich =5, zu identifizieren. Um sich das alles zu ersp aren, genügt es, die RESET-Taste zu betätigen.

Die Anweisung
RECALLOUS=234
wird zur Fehlermeldung ? OUT OF DATA ERROR führen, wenn nicht ein Bereich definiert worden ist, dessen Name mit den Zeichen OU beginnt. In einem solchen Falle wird APPLE_ SOFT unendlich lange auf einen vom Kassettenrecorder eingespielten Bereich warten. Sie können nur durch das Drücken der RESET-Taste den Computer wieder in den Griff bekommen.

KAPITEL 6

Die Input- und Output-Befehle

(Siehe auch Kapitel 3 unter "LOAD und SAVE" sowie Kapitel 5 unter "STORE und RE - CALL")

INPUT def

INPUT [string;] var [{, var}]

Wenn die wahlfreie Gruppe ausgelassen wird, dann wird durch INPUT ein Fragezeichen ausgedruckt. Danach muß der Benutzer eine Zahl (wenn var eine arithmetische Variable ist) oder Zeichen (wenn var eine Gruppenvariable ist) eingeben. Der Wert dieser Zahl oder der Gruppe wird in var eingesetzt.

Wenn die Gruppe vorhanden ist, dann wird sie genau wie festgelegt gedruckt. Nach der Gruppe erscheinen dann keine Fragezeichen, Leerräume oder anderen Elemente der Zeichensetzung (Anmerkung: Zeichensetzung hier im gramm: Sinn). Beachten Sie, daß nur eine wahlfreie Gruppe benutzt werden kann. Sie muß unmittelbar nach dem "INPUT" folgen, anschließend an die Gruppe muß ein Semikolon stehen.

Als numerische Eingabe ist nur eine reelle oder ganze Zahl zulässig, aber keine arithmetischen Ausdrücke. Die Zeichen Freiraum, +, - und E sowie die Periode sind rechtmäßige Teile einer numerischen Eingabe. Bei INPUT werden alle diese Zeichen oder irgendwelche Verkettungen in der entsprechenden Form akzeptiert. (Beispielsweise ist +E- akzeptabel, +- aber nicht.) Eine solche Eingabe allein wird mit Ø berechnet.

Bei numerischen Eingaben werden die Leerräume auf allen Positionen ignoriert. Wenn es sich bei der numerischen Eingabe nidht um eine reelle oder ganze Zahl, ein Komma oder ein Semikolon handelt, wird die Meldung

? REENTER

angezeigt, und die INPUT-Anweisung wird neu ausgeführt.

Wenn man ONERR GOTO benutzt, wobei ein anderes GOTO im Fehlerbearbeitungsprogramm das Programm zum Rücksprung auf die fehlerhafte Eingabeanweisung veranlaßt, kann es vorkommen, daß beim 86. INPUT-Fehler das Programm zum Monitor springt. Zur Regenerierung benutzen Sie dann reset ctrl C return. Diesem Problem kann man begegnen, indem man RESUME benutzt, um zum INPUT-Befehl zurückzugehen.

Ähnlich verhält es sich bei der Antwortzuordnung. So muß eine Antwort des Benutzers, die einer Gruppenvariable zugeordnet wird, immer eine einzelne Gruppe oder ein Literal sein,

und nicht ein Gruppenausdruck. Die dem ersten Zeichen vorausgehenden Leerstellen werden ignoriert. Wenn die Antwort eine Gruppe ist, würde ein Anführungszeichen irgendwo innerhalb der Gruppe die Meldung ?REENTER auslösen. Alle anderen Zeichen in einer Gruppe werden, jedoch mit Ausnahme der Anführungszeichen, von dtrl X und ctrl M, als zur Gruppe gehörige Zeichen anerkannt. Dazu gehören auch der Doppelpunkt und das Komma. Leerstellen im Anschluß an das letzte Ausführungszeichen werden nicht berücksichtigt.

Wenn es sich bei der Antwort um ein Literal handelt, dann werden die Anführungsstriche in in jedem Teil des Literals als Zeichen anerkannt. Davon ausgenommen ist das erste Zeichen, das kein Leerzeichen ist. Die Leerzeichen im Anschluß an das letzte Zeichen werden als Teil des Literals anerkannt. Davon ausgenommen sind jedoch das Komma und der Doppelpunkt (und ctrl X und ctrl M).

Wenn der Benutzer einfach die RETURN-Taste betätigt, wenn eine numerische Antwort erwartet wird, dann wird gemeldet: ? REENTER.
Die INPUT-Anweisung wird neu ausgeführt. Wenn nur die RETURN-Taste betätigt wird, wenn eine Antwortgruppe erwartet wird, dann wird die Antwort als Nullgruppe ausgelegt, und das Programm läuft weiter ab.

Nacheinanderfolgende Variable erhalten nacheinandereingegebene Werte. Gruppenvariable und arithmetische Variable können gemischt im gleichen INPUT-Befehl verwendet werden, aber die Antwort des Benutzers muß für jede von der entsprechenden Art sein. Die eingegebenen Antworten können mit Kommas oder durch returns getrennt werden. Wenn dann der Benutzer Kommas in einer Antwort eingibt, die nicht mit den Anführungsstrichen eingeleitet wurde, werden folglich die Kommas als Antworttrenner ausgelegt. Das trifft auch zu, wenn nur eine Antwort erwartet wird.

Wenn in einer INPUT-Antwort, die nicht durch ein Anführungszeichen eingeleitet wird, ein Doppelpunkt eingetastet wird, dann werden alle Zeichen, die danach eingegeben werden, nicht berücksichtigt. Nach einem Doppelpunkt werden auch Kommas ignoriert, so daß der Beginn einer weiteren Antwort durch das Betätigen der RETURN-Taste signalisiert werden muß. Wenn ein return vorkommen sollte, ehe allen var eine Antwort zugeordnet wurde, erscheinen auf dem Bildschirm zwei Fragezeichen, um anzuzeigen, daß noch eine weitere Antwort erwartet wird. Stößt der Computer auf ein return , wenn die Antwort mehr Ant-

wortfelder enthält als die Anweisung erwarten ließ oder falls ein Doppelpunkt in der letzten erwarteten Antwort vorkommt (aber nicht in einer Gruppe), dann wird die Meldung

? EXTRA IGNORED

angezeigt, und das Programm läuft weiter.

Beginnt eine INPUT-Antwort mit einem Doppelpunkt oder einem Komma, dann wird sie als Nullgruppe oder als Null gerechnet. Beachten Sie, daß in einem INPUT-Befehl die wahlfreie Gruppe ein Semikolon nach sich ziehen muß, Variable aber durch ein Komma getrennt werden müssen.

ctrl C kann eine INPUT-Anweisung unterbrechen, aber nur wenn es das erste eingegebene Zeichen ist. Das Programm wird angehalten, wenn man return eingibt. Der Versuch, den Programmablauf mit Hilfe von CONT wieder aufnehmen zu lassen, nachdem es wie oben beschrieben angehalten wurde, führt zur Fehlermeldung:

? SYNTAX ERROR.

Wenn es sich bei ctrl C nicht um das zuerst eingegebene Zeichen handelt, dann wird es wie jedes beliebige Zeichen behandelt. Wenn man versucht, den INPUT-Befehl im Sofortausführungsbetrieb zu verwenden, wird die Fehlermeldung

? ILLEGAL DIRECT ERROR

ausgelöst.

GET nur def
GET var

Damit wird ein einzelnes Zeichen vom Tastenfeld geholt, ohne es auf dem Bildschirm anzuzeigen. Bei Benutzung von GET braucht nicht die RETURN-Taste betätigt zu werden. Das Verhalten bei GET var überrascht manchmal:

ctrl @ gibt das Nullzeichen wieder. Das Ergebnis einer GET-Operation bei einem Linkspfeil oder bei ctrl H kann auch gedruckt werden, als wäre das Nullzeichen wiedergegeben worden. ctrl C wird behandelt wie jedes beliebige andere Zeichen. Es un-

terbricht nicht den Programmablauf.

Während APPLESOFT nicht gedacht ist, um durch GET-Operationen Weite für arithmetische Variablen zu holen, und auch nicht dafür ausgelegt ist, können Sie wohl GET avar benutzen, was aber an die folgenden strengen Beschränkungen gebunden ist:

ACHTUNG!

Holt man mit GET einen Doppelpunkt oder ein Komma, so führt das zur Fehlermeldung ? EXTRA IGNORED, der die Wiedergabe einer Null als des eingegebenen Wertes folgt. Das Plus- und Minuszeichen sowie ctrl @ , E , Leerraum und die Periode werden alle als Null wiedergegeben anstelle des eingegebenen Wertes. Die Eingabe von return oder eines nichtnumerischen Inputs führt zur Fehlermeldung ? SYNTAX ERROR.
Im Zusammenhang mit ONERR GOTO ...RESUME lassen zwei aufeinanderfolgende GETs das System solange im Schwebezustand, bis die RESET-Taste betätigt wird. Wenn GOTO das RESUME ersetzt, dann läuft alles bis zum 43. GET-Fehler (bei beliebiger Reihenfolge) gut, denn dort springt das Programm zum Monitor. Zur Regenerierung benutzen Sie die Eingabe: reset ctrl C return.

Wegen dieser Beschränkungen wird den verantwortungsvollen Programmierern empfohlen, mit
GET
Zahlen zu holen, indem Sie GET swar benutzen, und dann die sich ergebende Gruppe mit der VAL-Funktion in eine Zahl umzuwandeln.

DATA nur def
DATA [Literal / Gruppe / reelle Zahl / ganze Zahl][{, [Literal / Gruppe / reelle Zahl / ganze Zahl]}]

Mit dieser Anweisung wird eine Liste von Elementen aufgestellt, die für eine READ-Anweisung genutzt werden können. In der Reihenfolge der Befehle auf den numerischen Zeilen werden die Elemente aus den DATA-Anweisungen der Liste von Elementen hinzugefügt, die durch die im Programm vorausgehenden (auf den niedrigsten Zeilen stehenden) DATA-Anweisungen bereits zusammengekommen sind.

Die DATA-Anweisung muß nicht der READ-Anweisung im Programm vorausgehen, sondern

kann überall im Programm auftauchen.

DATA-Elemente, die in arithmetische Variable hineingelesen werden, entsprechen im allgemeinen den gleichen Regeln, wie sie für die Zuordnung von INPUT-Antworten zu arithmetischen Variablen gelten. Der Doppelpunkt kann jedoch nicht in einem numerischen DATA-Element als Zeichen enthalten sein.

Sollte ctrl C ein DATA-Element sein, dann stoppt es das Programm nicht, auch wenn es das erste Zeichen des Elements sein sollte. Mit dieser Ausnahme befolgen die DATA-Elemente, die in eine Gruppenvariable hineingelesen werden, die gleichen Regeln, wie sie für die Zuordnung von INPUT-Antworten zu Gruppenvariablen gelten.

> Es können entweder Gruppen oder Literale oder aber beides verwendet werden. Die Leerstellen vor dem ersten Zeichen der Gruppe und hinter der Gruppe werden nie beachtet. Jedes Anführungszeichen, das innerhalb einer Gruppe vorkommt, löst die Fehlermeldung "?SYNTAX ERROR" aus, aber alle anderen Zeichen werden als Zeichen innerhalb derGruppe akzeptiert, auch der Doppelpunkt und das Komma. (ctrl X und c.trl M aber nicht).
> Wenn das Element ein Literal ist, dann wird das Anführungszeichen als gültiges Zeichen überall in dem Literal akzeptiert, mit Ausnahme allerdings des ersten Zeichens das kein Leerraum ist. Der Doppelpunkt, das Komma, ctrl X und ctrl M werden nicht akzeptiert.

Siehe Abschnitt "INPUT" zu näheren Angaben.

Die DATA-Elemente können jede beliebige Mischung aus reellen und ganzen Zahlen, aus Gruppen und Literalen sein. Wird mit einer READ-Anweisung versucht, ein DATA-Element, das eine Gruppe oder ein Literal ist, einer arithmetischen Variablen zuzuordnen, dann wird die Fehlermeldung ? SYNTAX ERROR für die entsprechende DATA -Zeile angezeigt.

Wenn die Liste der Elemente in einer DATA-Anweisung ein "nichtexistentes" Element enthält, wird eine Null (numerische) oder eine Nullgruppe für dieses Element wiedergegeben. Das hängt aber von der Variablen ab, der das Element zugeordnet wird. Ein "nichtexistentes" Element kommt in einer DATA-Anweisung vor, wenn eine der nachfolgend aufgeführten Bedingungen erfüllt ist:

1. Es gibt kein Zeichen zwischen DATA und return (Leerräume nicht gezählt).

2. Das Komma ist das erste Zeichen, das dem DATA ohne Zwischenraum folgt.

3. Es gibt kein vom Leerraum verschiedenes Zeichen zwischen zwei Kommas.

4. Das Komma ist das letzte vom Leerraum verschiedene Zeichen vor dem return.

Also können, wenn die Anweisung 1ØØ DATA, , gelesen wird, bis zu drei Elemente, welche aus Nullen oder aus Nullgruppen bestehen, wiedergegeben werden. Wenn DATA in der Sofortausführung verwendet wird, löst es nicht eine SYNTAX-ERROR-Anzeige aus, sondern die DATA-Elemente sind für eine READ-Anweisung einfach nicht verfügbar.

READ imm & def
READ var [{, var}]

Wenn das erste READ in einem Programm ausgeführt wird, dann nimmt die erste Variable den Wert des Elements auf der DATA-Liste an (die DATA-Liste enthält alle Elemente aller DATA-Anweisungen eines gespeicherten Programms). Die zweite Variable (sofern eine solche vorhanden ist) nimmt den Wert des zweiten Elements der DATA-Liste an, usw. Wenn die Ausführung der READ-Anweisung beendet ist, wird hinter dem letzten verwendeten DATA-Element ein DATA-Listenzeiger gesetzt. Bei der nächsten READ-Anweisung, die ausgeführt wird, werden dann die Elemente der DATA-Liste, die sich hinter dem Listenzeiger befindet, verwendet. Entweder RUN oder RESTORE bringen den Zeiger wieder zurück zum ersten Element der Liste.

Wenn man versucht, mehr Daten lesen zu lassen, als auf der Liste sind, dann wird die Fehlermeldung ? OUT OF DATA ERROR IN linenum angezeigt, wobei linenum die Nummer der Zeile ist, auf der mit dem READ-Befehl zusätzliche Daten gelesen werden sollten.

In der Sofortausführung können Sie nur Elemente von DATA-Anweisungen lesen, die als Zeichen in dem gespeicherten Programm vorhanden sind. Die Elemente von DATA eines gespeicherten Programms können selbst dann gelesen werden, wenn für das gespeicherte Programm kein RUN gekommen ist. Sollten keine DATA-Anweisungen gespeichert worden sein, dann wird die Fehlermeldung ? OUT OF DATA ERROR angezeigt. Die Ausführung eines Programms in der Sofortausführungsweise stellt den DATA-Listenzeiger nicht auf das erste Element der

DATA-Liste. Zusätzliche Daten, die nicht gelesen werden, sind in Ordnung.

RESTORE imm & def

Bei RESTORE gibt es keine Parameter oder Optionen. Mit dieser Anweisung wird lediglich der DATA-Listenzeiger (siehe auch READ- und DATA-Anweisungen) zurückbewegt auf den Anfang der DATA-Liste.

PRINT imm & def

PRINT [{expr}[{, | ; [{ expr}]}]][, | ;]

PRINT {;}

PRINT {;}

Das Fragezeichen (?) kann als eine Abkürzung für PRINT verwendet werden. Es wird als PRINT aufgelistet.

PRINT ohne Optionen sorgt für den Zeilenvorschub und das Umschalten auf die nächste Zeile auf dem Bildschirm. Wenn Optionen erfüllt werden, dann werden die Werte der Liste der bestimmten Ausdrücke gedruckt. Wenn weder ein Komma noch ein Semikolon den Abschluß der Liste bilden, wird der Zeilenvorschub und das Umschalten nach dem nächsten gedruckten Posten vorgenommen. Wenn ein Posten der Liste mit Komma steht, dann wird das erste Zeichen des nächsten Postens auf Position 1 der nächsten verfügbaren Tabellenspalte erscheinen.

Die erste Tabellenspalte umfaßt die ersten 16 Positionen des Textfeldes von links gezählt, d. h. Pos. 1 bis 16. Die zweite Tabellenspalte nimmt die nächsten 16 Positionen (Pos. 17 bis 32) ein und kann nur verwendet werden, wenn auf Pos. 16 nichts gedruckt wird. Die dritte Tabellenspalte besteht aus den letzten 8 Positionen (Pos. 33 bis 4Ø) und kann nur verwendet werden, wenn auf Pos. 24 bis 32 nichts gedruckt ist.

Die Breite des "Textfeldbandes" kann mit Hilfe von verschiedenen POKE-Befehlen verändert werden. (Siehe dazu Anlage J)

ACHTUNG!

Die Tabellenspalte 3 bei PRINT funktioniert nicht einwandfrei, wenn das Textschreibfeld auf eine Breite unterhalb von 33 eingestellt wird. HTAB kann auch PRINT veranlassen, ein erstes Zeichen außerhalb des Textfeldes anzuzeigen.

Wenn einem Posten auf der Liste ein Semikolon folgt, dann wird der nächste Posten mit dem ersten verkettet. Es wird im direkten Anschluß ohne Zwischenraum geschrieben. Posten, die ohne Zwischenkommas oder Semikolons aufgelistet sind, werden verkettet, wenn die Posten ohne Syntax-Probleme identifiziert werden können. Das läßt sich am besten am Beispiel zeigen:

```
A = 1  : B=2 : C=3 : C(4)=5  : C5=7
PRINT 1/3 (2*4)51  ,        : PRINT 1(A)2(B)3C(4)C5
 .333333333851               1122357
 PRINT 3.4.5.6.  ,          : PRINT A."B."C.4
 3.4.5.6Ø                     1ØB.3.4
```

Bei PRINT muß sehr viel aufgewandt werden um festzustellen, was Sie wollen. Wenn zum Beispiel eine Periode nicht als Komma identifiziert werden kann, dann wird sie wie die Zahl Ø behandelt, wie es im oben angeführten Beispiel deutlich wird.

Wenn dem PRINT eine Reihe von Semikolons folgt, dann wird nur PRINT ausgeführt. Das ist zulässig. Wenn dem PRINT eine Reihe von Kommas folgt, dann wird pro Komma ein Zeilenvorschub bis zur nächsten Tabellenspalte vorgenommen. Das geht bis zur Grenze von 239 Zeichen pro Anweisung.

PRINT A$+B$

führt zur Fehlermeldung ? STRING TOO LONG ERROR, wenn die Länge der verketteten Gruppen 255 überschreitet. Allerdings kann man auch die offensichtliche Verkettung PRINT A$ B$ drucken lassen, ohne über die Länge besorgt zu sein.

IN# imm & def

IN# aexpr

Wählt die Eingabe über den Eingang aexpr . Er wird benutzt, um festzulegen, welches externe Gerät die Eingabe für die nachfolgenden INPUT-Anweisungen liefern wird. Die externen Geräte müssen über die Eingänge 1 bis 7, wie es von aexpr angegeben wird, angeschlossen sein.

IN#Ø gibt an, daß die folgende Eingabe über das Tastenfeld und nicht über ein externes Gerät erfolgen wird. Der Ø-Eingang ist nicht vom APPLESOFT adressierbar bei Benutzung

eines externen Geräts. Wenn über den Eingang |aexpr| kein externes Gerät angeschlossen ist, dann wird das System im Schwebezustand gehalten. Zur Regenerierung benutzt man

reset ctrl C return .

Wenn |aexpr| kleiner als Ø ist oder größer als 255, dann wird die Fehlermeldung ? ILLEGAL QUANTITY ERROR angezeigt.

VERBOTEN!

Liegt |aexpr| im Bereich 8 bis 255, wird APPLESOFT auf unvorhersehbare Weise verändert. Siehe PR# zu ähnlichem Output-Transfer.

PR# imm &def

PR# aexpr

Mit PR# wird die Ausgabe an den Eingang | aexpr| überwiesen, wobei |aexpr| im Bereich von 1 bis einschließlich 7 liegen muß. PR# Ø gibt das Output auf den Bilschirm, nicht zum Eingang Ø. Wenn kein externes Gerät am festgelegten Eingang vorhanden ist, dann wird das System in einem Schwebezustand gehalten. Geben Sie reset ctrl C return ein, um es zu regenerieren.

Wenn |aexpr| kleiner als Ø oder größer als 255 ist, dann wird die Fehlermeldung ? ILLEGAL QUANTITY ERROR angezeigt.

VERBOTEN!

Wenn |aexpr| im Bereich 8 bis 255 liegt, dann wird APPLESOFT in unvorhersehbarer Weise verändert. Siehe IN# zu ähnlichem Input-Transfer.

LET imm & def

[LET] avar [Index] = aexpr

[LET] svar [Index] = sexpr

Der Variablenname auf der Linken wird dem Wert der Gruppe oder des Ausdrucks auf der Rechten zugeordnet. Das LET ist wahlfrei.

LET A = 2 und A = 2 sind äquivalent.

Die Fehlermeldung ? TYPE MISMATCH ERROR wird angezeigt, wenn Sie versuchen,

a) einen Gruppenvariablennamen zu einem arithmetischen Ausdruck oder

b) einen Gruppenvariablennamen zu einem Literal oder

c) den Namen einer arithmetischen Variablen zu einem Gruppenausdruck zu geben.

Sollten Sie versuchen, einen Namen einer arithmetischen Variablen zu einem Literal zu geben, dann wird durch APPLESOFT versucht, das Literal als einen arithmetischen Ausdruck zu identifizieren.

DEF def
FN imm & def

DEF FN Name (reelle avar) = aexpr1
FN Name (aexpr2)

Hiermit wird dem Benutzer gestattet, Funktionen in einem Programm zu definieren. Zuerst wird der Funktionsname FN mit Hilfe von DEF bestimmt. Wenn dann die Programmzeile, die die Funktion DEFiniert, ausgeführt wurde, kann die Funktion im weiteren Verlauf in der Form FN Name (Argument) benutzt werden, wobei das Argument aexpr2 irgendein arithmetischer Ausdruck sein kann. Der aexpr der DEFinition kann nur eine Programmzeile lang sein. Der bestimmte FN-Name kann immer dort benutzt werden, wo arithmetische Funktionen im APPLESOFT benutzt werden können.

Solche Funktionen können im Verlaufe des Programms neu definiert werden. Die Regeln für die Verwendung von arithmetischen Variablen treffen dann noch zu. Insbesondere die ersten beiden Zeichen eines Namens müssen einheitlich sein. Wenn diese beiden Zeilen

```
1Ø DEF FN ABC(I) = COS(I)
2Ø DEF FN ABT(I) = TAN(I)
```

ausgeführt werden, dann erkennt APPLESOFT die Definition einer Funktion FN AB in Zeile 1Ø. In der Zeile 2Ø wird die Funktion FN AB neu definiert.

In der DEF-Anweisung ist die reelle avar eine Pseudo-Variable. Wenn der vom Nutzer bestimmte Funktionsname FN später verwendet wird, wird er mit einem Argument aexpr2 abgerufen. Dieses Argument wird gegen die reelle avar immer dort ersetzt, wo es in dem aexpr1 der Definition auftritt. Der aexpr1 kann eine beliebige Anzahl von Variablen enthalten,

aber natürlich entspricht nur eine dieser Variablen (in den meisten Fällen) der Pseudo-Variablen, der reellen avar, und deshalb entspricht sie der Argument-Variablen.

Die reelle avar der Definition braucht nicht in aexpr1 zu erscheinen. In diesem Falle wird das Argument der Funktion bei der Berechnung von aexpr1 ignoriert, wenn die Funktion zu einem späteren Zeitpunkt im Programm verwendet wird. Selbst in diesem Fall wird aber das Argument der Funktion selbst berechnet, es muß demzufolge etwas Zulässiges sein.

Beispiel:

```
100 DEF FN A(W) = 2 * W + W
110 PRINT FN A (23)
120 DEF FN B(X) = 4 + 3
130 G = FN B(23)
140 PRINT G
150 DEF FN A(Y) = FN B(Z) + Y
160 PRINT FN A(G)

RUN
69          [FN A(23)=2 * 23+23]
7           [FN B(anything)=7]
14          [new FN A(7)=7+7]
```

Wenn im verzögerten Ausführungsbetrieb eine DEF FN Namen-Anweisung nicht vor der Verwendung des FN Namens ausgeführt wird, dann erscheint die Fehlermeldung ? UNDEF'D FUNCTION ERROR auf der Anzeige.

Gruppenfunktionen, die vom Nutzer bestimmt werden, sind nicht zulässig. Definierte Funktionen, die einen Namen einer ganzen Zahl (name%) als Namen oder als reelle avar benutzen, sind nicht zulässig.

Wenn eine neue Funktion durch eine DEF-Anweisung definiert wird, dann werden 6 Byte des Speichers verbraucht, um den Zeiger für diese Definition zu speichern.

KAPITEL 7

Befehle zum Steuerungsablauf

GOTO imm & def

GOTO linenum

Dieser Befehl verzweigt zur Zeile mit Zeilennummer linenum. Sollte eine solche Zeile nicht vorhanden oder linenum nicht im GOTO-Befehl enthalten sein, wird die Fehlermeldung ? UNDEF'D STATEMENT ERROR IN linenum angezeigt, wobei linenum die Nummer der Zeile ist, in der sich die GOTO-Anweisung befindet.

IF imm & def

IF expr THEN Anweisung [{ : Anweisung}]

IF expr THEN [GOTO] linenum

IF expr [THEN] GOTO linenum

Wenn expr ein arithmetischer Ausdruck ist, dessen Wert = Ø (und dessen absoluter Wert größer ist als etwa 2.93873E-39), dann wird \expr\als wahr betrachtet, und jede Anweisung bzw. alle Anweisungen nach dem THEN wird bzw. werden ausgeführt.

Wenn expr ein arithmetischer Ausdruck ist, dessen Wert = Ø (oder dessen absoluter Wert kleiner ist als 2.93873E-39), dann wird jede Anweisung nach dem THEN ignoriert. Die im Programm folgende Zeile wird dann ausgeführt.

Wenn eine IF-Anweisung in einem Programm der Sofortausführung vorkommt und \expr\= Ø, dann wird APPLESOFT das gesamte verbleibende Programm unberücksichtigt lassen. Wenn expr ein arithmetischer Ausdruck ist, an dem Stringausdrücke oder logische Operatoren für Strings beteiligt sind, wird expr berechnet, indem die alphabetische Rangfolge der Stringausdrücke, wie sie durch die ASCII-Codes für die beteiligten Zeichen (siehe Anlage K) bestimmt sind, verglichen wird.

Die Anweisungen in der Form IF expr THEN sind gültig. Es wird keine Fehlermeldung angezeigt. Ein THEN ohne ein dazugehöriges IF oder ein IF ohne ein THEN lösen die Fehlermeldung ? SYNTAX ERROR aus.

APPLESOFT wurde nicht ausgelegt oder ist nicht dazu gedacht, daß der expr im IF-Befehl ein Stringausdruck sein kann. Stringvariable und Strings können aber als expr unter fol-

genden strikten Bedingungen benutzt werden:

Wenn expr irgendein Stringausdruck ist, dann ist \ expr\ von Null verschieden, selbst wenn expr eine Stringvariable ist, der kein Wert zugeordnet wurde, oder "Ø" oder die Nullstring '''', zugeordnet wurde. Die Literalnull wie in IF '''' THEN... wird als Null berechnet.

ACHTUNG!

IF String THEN... löst, wenn es mehr als dreimal in einem gegebenen Programm benutzt wird, die Fehlermeldung ? FORMULA TOO COMPLEX ERROR aus.

ACHTUNG!

Wenn expr eine Stringvariable sein sollte,und die vorausgehende Anweisung ordnete den Nullstring irgendeiner Stringvariable zu , dann wird \ expr\ mit Null berechnet. Beispielsweise schreibt das Programm

```
12Ø IF AØ  THEN PRINT "AØ"
13Ø IF BØ  THEN PRINT "BØ"
14Ø IF XØ  THEN PRINT "XØ",
```

wenn der RUN-Befehl kommt,

```
AØ
BØ
XØ
```

aus, da die Strings AØ, BØ und XØ mit ungleich Null berechnet werden. Doch wenn man die Zeile 1ØØ QØ = "" hinzufügt, dann werden alle drei Strings mit Null berechnet, und es wird kein Output gedruckt. Das Löschen der Zeile 1ØØ oder das Hinzufügen fast jeder beliebigen Zeile 11Ø, wie beispielsweise 11Ø F = 3 verursacht, daß alle drei Strings wieder mit ungleich Null berechnet werden.

ACHTUNG!

Vor dem THEN werden durch den Buchstaben A Identifizierungsprobleme verursacht. Bei IF BETA THEN 23Ø wird identifiziert: IF BET AT HEN23Ø. Das wiederum löst die Fehlermeldung ? SYNTAX ERROR aus, wenn die Zeile ausgeführt werden soll.

Das folgende ist identisch miteinander:

```
IF A=3 THEN 16Ø
IF A=3 GOTO 16Ø
IF A=3 THEN GOTO 16Ø
```

FOR imm & def

FOR reelle avar = aexpr1 TO aexpr2 [STEP aexpr3]

\avar\ wird auf \aexpr1\ gesetzt, und die Anweisungen nach dem FOR werden solange ausgeführt, bis die Anweisung NEXT avar vorkommt, wo avar der gleiche Name ist wie in der FOR-Anweisung. Dann wird \avar\ um \aexpr3\ (wenn \aexpr3\ nicht festgelegt, dann 1) erhöht. Als nächstes wird \avar\ mit \aexpr2\ verglichen, und wenn \avar\ > \aexpr2\ dann setzt sich der Programmablauf mit der dem NEXT folgenden Anweisung fort. Wenn \avar\ < = \aexpr2\ , setzt sich die Ausführung bei der Anweisung nach dem FOR fort.

Wenn \aexpr3\ < Ø, dann verändert sich die Ausführung geringfügig, nachdem \aexpr3\ zum \avar\ hinzugezählt wurde. Wenn \avar\ < \aexpr2\ , dann setzt sich die Ausführung des Programms mit der Anweisung nach NEXT fort. Wenn \avar\ > = \aexpr2\ , dann setzt sich die Ausführung des Programms mit der Anweisung nach dem FOR fort.

Die arithmetischen Ausdrücke, die die Parameter der FOR-Schleife bilden, können reelle Zahlen, reelle Variablen, ganze Zahlen oder ganzzahlige Variablen sein. Die reelle avar muß jedoch eine reelle Variable sein. Der Versuch, eine ganzzahlige Variable als reelle avar zu nehmen, führt zur Fehlermeldung ? SYNTAX ERROR .

Da avar erst auf der Basis der FOR...NEXT - Schleife erhöht und mit aexpr2 verglichen wird, wird der Programmabschnitt innerhalb der Schleife jedenfalls mindestens einmal ausgeführt.

FOR...NEXT - Schleifen dürfen sich nicht überschneiden. Falls es dazu kommen sollte, wird die Fehlermeldung ? NEXT WITHOUT FOR ERROR Sie davon informieren. Wenn die FOR-Schleifen mehr als zehn Ebenen tief versch achtelt werden, kündigt das die Fehlermeldung:? OUT OF MEMORY ERROR an.

Wenn man in einem Programm bei Sofortausführung eine FOR...NEXT-Schleife laufen lassen will, sollte sich sowohl die FOR- als auch die NEXT-Anweisung auf der Zeile mit der gleichen Zeilennummer befinden (eine Zeile ist bis zu 239 Zeichen lang).

ACHTUNG!

Wenn der Buchstabe A direkt vor dem TO verwendet wird, muß darauf geachtet werden, daß zwischen T und O kein Zwischenraum bleibt. Die Anweisung "FOR I=BETA TO 56" ist korrekt, aber "FOR I=BETA T 0 56" wird als "FOR I=BET AT 056" identifiziert. Bei der Ausführung erscheint dann die Fehlermeldung ? SYNTAX ERROR auf der Anzeige.

Jede aktive FOR...NEXT-Schleife verbraucht 16 Byte Speicher.

NEXT imm & def

NEXT [avar]

NEXT avar [{,avar}]

Das NEXT ist die Basis der FOR...NEXT-Schleife. Wenn NEXT auftritt, wird es vom Programm entweder außer Acht gelassen oder das Programm verzweigt zu der Anweisung, die im dazugehörigen FOR festgelegt ist. Das ist abhängig von den Bedingungen, die im Abschnitt "FOR" erläutert wurden.

Die Mehrfach-avars müssen in der richtigen Reihenfolge festgelegt werden, da die FOR... NEXT-Schleifen ineinander verschachtelt sind und sich nicht überschneiden dürfen. Falsch angeordnete avars werden zur Fehlermeldung ? NEXT WITHOUT FOR ERROR führen.

Eine NEXT-Anweisung, in der kein Variablenname angegeben ist, ist nicht hinreichend für die letzte aktive FOR-Schleife. Wenn keine FOR-Anweisung mit dem gleichen Variablennamen wirksam ist oder wenn keine FOR-Anweisung mit irgendeinem Namen wirksam ist, und ein namenloses NEXT tritt auf, dann wird die Fehlermeldung ? NEXT WITHOUT ERROR

geschrieben. NEXT ohne avar wird schneller ausgeführt als NEXT mit avar.

In der Betriebsart Sofortausführung sollten die FOR-Anweisung und die dazugehörige NEXT-Anweisung in einer Zeile ausgeführt werden. Wenn noch ein FOR aus dem verzögerten Ausführungsbetrieb wirksam ist, kann eine NEXT-Anweisung im Sofortausführungsbetrieb einen Sprung in das verzögerte Ausführungsprogramm auslösen, wo das geeignet erscheint. Doch wenn eine FOR-Anweisung in der Sofortausführung erfüllt wurde, wird eine NEXT-Anweisung in einer anderen Zeile des Sofortausführungsprogramms die Anzeige von ? SYNTAX ERROR auslösen, es sei denn, es gibt keine Zeilen dazwischen und das NEXT steht allein und ohne Namen.

Beispiel:

```
]FOR I=1 TO 5 : PRINT I
 1
]NEXT
 2
]NEXT
 3
]NEXT I
 ? SYNTAX ERROR IN  xxxx     (xxxx irgendeine Zeilennummer)
```

GOSUB   imm & def
GOSUB   linenum

Das Programm verzweigt in die angegebene Zeile. Wenn eine RETURN-Anweisung ausgeführt wird, verzweigt das Programm zu der Anweisung, die dem zuletzt ausgeführten GOSUB folgt. Jedesmal wenn ein GOSUB ausgeführt wird, wird die Adresse der nachfolgenden Anweisung an der obersten Stelle des Stacks dieser Adressen gespeichert, so daß das Programm später den Weg zurück findet. Jedesmal wenn ein RETURN oder ein POP ausgeführt wird, wird die oberste Adresse im RETURN-Stack entfernt.

Wenn die angegebene Zeilennummer nicht mit einer existierenden Programmzeilennummer übereinstimmt, dann wird ? UNDEF'D STATEMENT ERROR IN linenum angezeigt, wobei linenum die Nummer der Programmzeile angibt, in der die GOSUB-Anweisung enthalten ist. Der Teil der Fehlermeldung " In linenum" wird ausgelassen, wenn das GOSUB in der Sofortausführung verwendet wird.

Wenn die GOSUBs bis über 25 Ebenen tief verschachtelt sind, wird die Fehlermeldung ? OUT OF MEMORY ERROR angezeigt. Jedes aktive GOSUB (eines, für das noch kein RETURN kam) belegt 6 Byte im Speicher.

RETURN imm & def
RETURN

Für dieses Kommando gibt es keine Parameter oder Optionen. Es beinhaltet eine Verzweigung zu der Anweisung, die dem letzten GOSUB unmittelbar folgt. Die Adresse der verzweigten Anweisung ist die oberste des RETURN-Stacks (siehe GOSUB und POP). Wenn in einem Programmablauf die RETURN-Anweisung einmal öfter als die GOSUB-Anweisung vorkommt, dann wird die Fehlermeldung ? RETURN WITHOUT GOSUB ERROR angezeigt.

POP imm & def
POP

Mit POP sind keine Parameter oder Optionen verbunden. Ein POP erzielt die gleiche Wirkung wie ein RETURN ohne Verzweigung. Durch das nächste RETURN wird dann eine Verzweigung zum vorletzten GOSUB erfolgen, statt zur Anweisung zu verzweigen, die nach dem letzten GOSUB steht. Es wird deshalb "POP" bezeichnet, weil es eine Adresse aus der obersten Position des RETURN-Adressen-Stacks wegsteckt (engl. to pop).

Wenn POP ausgeführt wird, bevor ein GOSUB vorgekommen ist, dann wird die Fehlermeldung ? RETURN WITHOUT GOSUB ERROR angezeigt, weil es keine RETURN-Adressen im Stack gibt.

ON...GOTO def

ON...GOSUB def

ON aexpr GOTO linenum {[,linenum]}

ON aexpr GOBUS linenum {[,linenum]}

Der Befehl ON...GOTO verzweigt zur Zeilennummer, die von dem \ aexpr\ -ten Posten der Liste von Zeilennummern, die dem GOTO folgen, bestimmt wird. ON...GOSUB verfährt in ähnlicher Weise, aber es wird ein GOSUB und kein GOTO ausgeführt.

Wenn \ aexpr\ gleich Null oder größer als die Zahl der aufgelisteten Alternativzeilennummern, jedoch kleiner als 256 ist, dann fährt das Programm mit der Ausführung der nächsten Anweisung fort.

\ aexpr\ muß sich in dem Bereich von Ø bis 255 bewegen, um die Fehlermeldung ? ILLEGAL QUANTITY ERROR zu vermeiden.

ONERR GOTO nur def

ONERR GOTO linenum

Sollte ein Fehler auftreten, kann ONERR GOTO verwendet werden, um die Anzeige einer Fehlermeldung und das Anhalten des Programmlaufs zu vermeiden. Das Kommando setzt ein Kennzeichen, das einen unbedingten Sprung (falls der Fehler im Programm weiter unten auftritt) zur Zeilennummer des Programms, die durch linenum bestimmt wird, zur Folge hat. POKE 216, Ø löscht das Fehlersuchkennzeichen, so daß die normalen Fehlermeldungen ausgedruckt werden.

Die ONERR-GOTO-Anweisung muß ausgeführt werden, ehe sich ein Fehler einstellt, damit das Programm nicht unterbrochen wird. Wenn es in einem Programm zu einem Fehler kommt, dann wird der Code für die Art des Fehlers auf dem Dezimalspeicherplatz 222 gespeichert. Wollen Sie prüfen, welcher Fehler vorgekommen ist, dann geben Sie ein:

```
PRINT PEEK(222)
```

Code	Fehlermeldung	Code	Fehlermeldung
Ø	NEXT without FOR	12Ø	Redimensioned Array
16	Syntax	133	Division by Zero
22	RETURN without GOSUB	163	Type Mismatch
42	Out of DATA	176	String Too Long
53	Illegal Quantity	191	Formula Too Complex
69	Overflow	224	Undefined Function
77	Out of Memory	254	Bad Response to INPUT Statement
9Ø	Undefined Statement	255	Ctrl C Interrupt Attempted
1Ø7	Bad Subscript		

ACHTUNG!

Besondere Vorsicht muß walten bei der Bearbeitung von Fehlern in FOR...NEXT-Schleifen oder zwischen GOSUB und RETURN, da die Zeiger und RETURN-Stacks durcheinanderkommen können. Das Fehlerbearbeitungsmaschinenprogramm muß die Schleife neu starten, wobei zum FOR-oder GOSUB-Befehl zurückgegangen werden muß,und nicht zum NEXT- oder RETURN-Befehl. Nach der Fehlerkorrektur wird ein return zum NEXT oder zum RETURN die entsprechende Meldung auslösen: ? NEXT WITHOUT FOR ERROR oder ? RETURN WITHOUT GOSUB ERROR

ACHTUNG!

Wenn ONERR GOTO mit RESUME zur Fehlerbearbeitung in GET-Anweisungen verwendet wird, bleibt das Programm im "Schwebezustand", wenn zwei nacheinanderfolgende GET-Fehler auftreten und kein erfolgreiches GET dazwischen liegt. Um aus dieser Situation wieder herauszukommen, verwenden Sie reset ctrl C return. Wenn das GOTO das Maschinenprogramm zur Fehlerbearbeitung beendet, dann läuft alles bestens (siehe aber nächstes ACHTUNG!)

ACHTUNG!

Wenn ONERR im TRACE-Betrieb (Protokollprogramm) oder in einem Programm, welches eine

PRINT-Anweisung enthält, benutzt wird, dann verursacht es einen Sprung zum Monitor nach 43 Fehlern. Wo diese Fehler durch eine INPUT-Anweisung hervorgerufen wurde, läuft alles reibungslos, wenn RESUME mit eingesetzt wird. Doch wenn GOTO das Fehlerbearbeitungsmaschinenprogramm beendet, dann kommt es nach dem 87. INPUT-Fehler zu einem Sprung zum Monitor. Mit reset ctrl C return kommen Sie wieder zu APPLESOFT.

Wenn Sie mit irgendeinem der eben erläuterten Probleme nicht zurechtkommen, führen Sie einfach ein CALL zu dem folgenden assembler-sprachlichen Unterprogramm als Teil Ihres Fehlerbearbeitungsunterprogramms aus. In den Monitor geben Sie die Hexa-Daten ein:

68 A8 68 A6 DF 9A 48 98 48 6Ø

oder in APPLESOFT geben Sie die De zimaldaten ein:

1Ø4 168 1Ø4 166 223 154 72 152 72 96

Beispielsweise könnten Sie in APPLESOFT mit POKE die Dezimalzahlen in die Speicherplätze 768 bis 777 geben. Danach könnten Sie CALL 768 in Ihrem Unterprogramm zur Fehlerbearbeitung benutzen.

RESUME def
RESUME

Wenn es am Schluß des Fehlerbearbeitungsunterprogrammes benutzt wird, löst es den weiteren Programmlauf, beginnend mit der Anweisung, in der der Fehler vorkam, wieder aus. Wenn ein RE SUME auftritt, bevor es zu einem Fehler kam, kann die Fehlermeldung ? SYNTAX ERROR IN 65278 angezeigt werden. Es kann aber auch zu eigenartigen Ereignissen kommen. Normalerweise wird das Programm angehalten oder es "hängt".

Wenn es zu einem Fehler in einem Unterprogramm zur Fehlerbearbeitung kommt, wird das Programm in eine endlose Schleife gesetzt, wenn RESUME kommt. Man verwendet reset ctrl C return, um die Situation zu bereinigen. In der Sofortausführung wird es das System in der "Schwebe" halten, eine SYNTAX-ERROR-Meldung auslösen oder die Ausführung eines bestehenden oder gar gelöschten Programms beginnen.

KAPITEL 8

Graphische Darstellungen und Spielsteuerung

TEXT imm & def

TEXT

Keine Parameter. Mit TEXT wird der Bildschirm auf den Textbetrieb über den gesamten Bildschirm (40 x 40) eingestellt. Die Umschaltung kann vom LR-Graphik-Betrieb oder von einer der beiden HR-Graphik-Betriebsarten erfolgen. Das Ladesymbol und der Cursor werden in die letzte Zeile des Bildschirms gesetzt. Bei der Eingabe von TEXT im Textbetrieb kommt es zum gleichen Ergebnis wie bei VTAB 24.

Eine Anweisung wie

175 TEXTILE=127

löst die Ausführung der durch das Tabuwort TEXT festgelegten Anweisung aus, ehe die Fehlermeldung

? SYNTAX ERROR

auf der Anzeige erscheint.

Wenn das Textschreibfeld auf irgendeine andere Größe eingestellt wurde (siehe Anlage J), dann erfolgt durch TEXT die Umschaltung auf Nur-Textanzeige.

GR imm & def

GR

Keine Parameter. Dieses Kommando stellt den LR-Graphik-Betrieb für die Anzeige (40 x 40) ein, wobei am unteren Rand der Anzeige vier Zeilen für Text freigehalten werden. Die Bildschirmfläche wird auf Schwarz gestellt und freigemacht. Der Cursor wird in das Textschreibfeld gesetzt. Das Ganze läßt sich auch auf Vollgraphik (40 x 48) umstellen, wenn man nach erfolgtem GR das Kommando

POKE -163Ø2, Ø

bzw. den äquivalenten Befehl

POKE 49234, Ø

eingibt. Wenn nach einem den Gesamtbildschirm betreffenden POKE-Befehl das GR kommt, wird dadurch wieder der Text/HR-Graphik-Parallelbetrieb eingestellt.

Nach dem GR-Befehl wird COLOR auf Null gesetzt.

ACHTUNG!

Sollte das Tabuwort (Befehlswort) GR als die ersten beiden Zeichen eines Variablennamens verwendet werden, kann der Fall eintreten, daß GR noch vor der Anzeige der Fehlermeldung

? SYNTAX ERROR

ausgeführt wird. Wenn also die Anweisung

GRIN=5

ausgeführt wird, bleibt der Bildschirm unerwarteter Weise schwarz.

ACHTUNG!

Wenn GR gegeben wird, während HGR wirksam ist, verhält es sich normal. Wenn es aber gegeben wird, wenn HGR2 wirksam ist, dann macht GR den gewöhnlich dafür vorgesehenen Speicher frei und zeigt Text- und LR-Graphikseite 2 an. Wenn Sie zum normalen Betrieb zurückschalten wollen, brauchen Sie nur TEXT einzugeben. Benutzen Sie in Programmen TEXT, bevor Sie von HGR2 auf GR umschalten.

COLOR imm & def
COLOR= aexpr
Mit diesem Befehl wird die Schreibfarbe im LR-Graphik-Betrieb eingestellt. Wenn \aexpr\ eine reelle Zahl ist, wird sie in eine ganze Zahl umgewandelt. Der Bereich der Werte für \ aexpr\reicht von 0 bis 255. Dieses sind bearbeitete Modulo 16.

Die Farbbezeichnungen und deren Zahlen lauten:

Ø - schwarz
1 - Magenta-Rot
2 - dunkelblau
3 - purpurrot
4 - dunkelgrün
5 - grau
6 - blau
7 - hellblau
8 - braun
9 - orange
10 - grau
11 - rosa
12 - grün
13 - gelb
14 - blaugrün
15 - weiß

Durch den GR-Befehl wird COLOR auf Null gesetzt.
Um COLOR für einen gegebenen Punkt auf dem Bildschirm herauszufinden, genügt es eine SCRN-Anweisung einzugeben.
Wenn COLOR im Textbetrieb verwendet wird, dann stellt es einen Faktor zur Bestimmung des Zeichens dar, das auf dem Bildschirm mit Hilfe einer PLOT-Anweisung abgebildet werden soll.

Wenn COLOR im HR-Graphik-Betrieb benutzt wird, findet es keine Berücksichtigung.

PLOT imm & def

PLOT aexpr1, aexpr2

Im LR-Graphik-Betrieb (zur graphischen Darstellung von Figuren bei geringer Bildauflösung) wird durch diesen Befehl ein Punkt mit der X-Koordinate \aexpr1\ und der Y-Koordinate \aexpr2\ abgebildet. Die Farbe des Punktes wird durch die zuletzt erteilte COLOR-Anweisung (COLOR=Ø, wenn nicht anders festgelegt) bestimmt.

\aexpr1\ muß sich im Bereich von Ø bis 39 befinden, und \aexpr2\ im Bereich von Ø bis 47. Andernfalls wird die Fehlermeldung

? ILLEGAL QUANTITY ERROR

abgebildet.

Wenn man während des Textbetriebes oder während des HR-Graphik/Text-Parallelbetriebs, wo dann \aexpr2\ im Bereich von 40 bis 47 liegt, versucht, PLOT anzuweisen, dann erscheint ein Zeichen dort, wo sonst der farbige Punkt hätte erscheinen müssen. (Ein Zeichen belegt den Raum für zwei LR-Graphikpunkte im Vertikalstack.)
Das Kommando bringt keine sichtbaren Folgen mit sich, wenn es im HR-Graphik-Betrieb HGR2 verwendet wird, selbst wenn ihm ein GR-Befehl vorausgeht, da der Bildschirm nicht auf den LR-Teil des Speichers (Seite 1) "blickt".

Der Ursprung (Ø, Ø) für alle graphischen Darstellungen befindet sich in der linken oberen Ecke des Bildschirms.

HLIN imm & def

HLIN aexpr1, aexpr2 AT aexpr3

Wenn dieser Befehl im LR-Graphik-Betrieb benutzt wird, dann wird durch ihn eine Linie von (\ aexpr1 \ , \ aexpr3 \ ) nach (\aexpr2\ , \aexpr3 \ ) gezogen. Sie erscheint in der Farbe, die durch die letzte COLOR-Anweisung vorgegeben ist.

\ aexpr1 \ und\aexpr2\ muß im Bereich von 0 bis 39, und \aexpr3\ im Bereich von 0 bis 47 liegen. Ansonsten erscheint die Fehlermeldung

? ILLEGAL QUANTITY ERROR

auf der Anzeige. \ aexpr1\ kann größer als, gleich oder kleiner als \ aexpr2 \ sein.

Wenn HLIN benutzt wird, während das System im Textbetrieb läuft oder im Graphik/Text-Betrieb läuft, wo \aexpr3\ im Bereich von 0 bis 47 liegt, wird eine Zeile Zeichen dort gedruckt, wo sonst die Zeile von Punkten hätte erscheinen müssen. (Ein Zeichen belegt den Raum für zwei LR-Graphikpunkte im Vertikalstack.)

Das Kommando zieht keine sichtbaren Folgen nach sich, wenn es im HR-Graphik-Betrieb verwendet wird.

Beachten Sie, daß das H in diesem Befehl sich auf "horizontal" und nicht auf "hohe Auflösung - HR" bezieht. Außer bei HLIN und HTAB bezieht sich das Präfix H sonst immer auf HR-Anweisungen.

VLIN imm & def

VLIN aexpr1, aexpr2 AT aexpr3

Dieses Kommando bewirkt im LR-Graphik-Betrieb das Ziehen einer Linie von (\ aexpr1 \ , \ aexpr3 \ ) nach (\aexpr2 \, \ aexpr3 \ ). Die Farbe wird vom letzten COLOR-Befehl vorgegeben.

\ aexpr1\ und \aexpr2\ müssen im Bereich von 0 bis 47, \ aexpr3\ im Bereich von 0 bis 39 liegen. Sonst wird die Fehlermeldung

? ILLEGAL QUANTITY ERROR

angezeigt. \ aexpr1\ kann größer als, gleich oder kleiner als \ aexpr2\ sein.

Wenn das System im Textbetrieb oder im Graphik/Text-Parallelbetrieb läuft, wo dann \aexpr2\ im Bereich von 40 bis 47 liegt, dann wird der Teil der Linie, der sich im Textfeld befindet, als eine Zeile von Zeichen erscheinen, wo normalerweise die Graphik-Punkte abgebildet hätten werden müssen.

Der Befehl hat keine sichtbaren Auswirkungen, wenn er im HR-Graphik-Betrieb verwendet wird.

SCRN imm & def

SCRN (aexpr1, aexpr2)

Im LR-Graphik-Betrieb gibt die SCRN-Funktion den Farb-Code des Punktes wieder, dessen X-Koordinate \ aexpr1\ und dessen Y-Koordinate \aexpr2\ ist.

ACHTUNG!

Obwohl im LR-Graphik-Betrieb die Punkte auf den Bildschirmpositionen (x, y) abgebildet werden, wobei x im Bereich 0 bis 39 und y im Bereich von 0 bis 47 liegt, werden aber von der SCRN-Funktion X-Werte zugelassen, die bei x und y im Bereich von 0 bis 47 liegen. Wenn nun in der SCRN-Funktion ein X-Wert (\aexpr1\) aus dem Bereich 40 bis 47 verwendet wird, dann gibt die ausgeworfene Zahl die Farbe des Punktes an, dessen X-Koordinate (\aexpr\ -40) und dessen Y-Koordinate (\aexpr2\+16) ist. Wenn sich (\aexpr2\ +16) in dem Bereich 39 - 47 befindet, dann wird im normalen Graphik/Text-Parallelbetrieb die ausgeworfene Zahl mit einem Textzeichen, das sich auf der angegebenen Position im Textbereich unterhalb der Graphik-Anzeige des Bildschirms befindet, in Beziehung stehen. Sollte (\aexpr2\+16) im Bereich von 48 bis 63 liegen, dann wird SCRN eine Zahl wiedergeben, die sich auf gar nichts auf dem Bildschirm bezieht.

Im Textbetrieb werden durch SCRN Zahlen aus dem Bereich 0 bis 15 angegeben, deren Wert durch

die oberen 4 Bit des Zeichens, falls aexpr2 ungerade, oder

die unteren 4 Bit des Zeichens, falls aexpr2 gerade,

ausgedrückt wird. Dieses Zeichen befindet sich auf Position:

(aexpr1+1, INT ((aexpr2+1)/2)).

Demzufolge wird der Ausdruck

CHR$(SCRN(X-1, 2*(Y-1))+16*SCRN(X-1, 2*(Y-1)+1)

das Zeichen auf der Position (x,y) wiedergeben.

Im HR-Graphik-Betrieb "blickt" man mit SCRN weiter auf die LR-Graphik-Anzeige, und die Zahl, die von der Funktion wiedergegeben wird, bezieht sich nicht auf die HR-Anzeige.

SCRN wird nur als Tabuwort identifiziert, wenn das nächste, sich ohne Leerzeichen anschließende Zeichen die nach rechts geöffnete Klammer ist.

HGR imm & def

HGR

Keine Parameter. Mit diesem Befehl wird der HR-Graphik-Betrieb des Bildschirms (Anzeige: 280 x 160) eingestellt, wobei am unteren Rand des Bildschirms vier Zeilen für Text verbleiben. Der Bildschirm wird freigemacht, und eine schwarze Bildfläche entsteht.
Seite 1 des Speichers (8K - 16K) wird angezeigt. HCOLOR-Anweisungen verändern sich durch diesen Befehl nicht. Der Speicher für den Textbetrieb wird nicht beeinflußt. Das HGR-Kommando läßt das Textschreibfeld auf dem gesamten Bildschirm bestehen, doch nur die untersten vier Zeilen sind unterhalb der Anzeige für graphische Darstellungen sichtbar. Der Cursor bleibt im Textfeld, doch er wird erst dann sichtbar, wenn er in eine der vier Zeilen am unteren Rand gesetzt wird.
Der Bildschirm kann auf Graphik-Anzeige über den gesamten Bildschirm (Anzeige: 280 x 192) umgestellt werden, indem man nach der Ausführung der HGR-Anweisung

POKE -16302, 0

eingibt oder

POKE 49234, 0 ,

das dem oberen gleich ist, verwendet. Wenn einem der beiden o.g. POKE-Befehle ein HGR folgt, dann wird wieder die HR-Graphik/Text-Parallelanzeige eingestellt.

ACHTUNG!

Sollte das Tabuwort HGR als Anfang eines Variablennamens benutzt werden, kann die HGR-Anweisung ausgeführt werden, ehe die Fehlermeldung

? SYNTAX ERROR

angezeigt wird. So ergibt die Ausführung einer Anweisung

HGRIP=4

eine unerwartete Umschaltung in die HR-Graphik-Betriebsweise, die Ihr Programm löschen kann.

ACHTUNG!

Ein sehr langes Programm, das sich bis über die Speicherstelle 8192 hinaus erstreckt, kann teilweise gelöscht werden, wenn Sie Ihr HGR eingeben. Es kann aber auch geschehen, daß das Programm in die HR-Graphik-Seite 1 "hineinschreibt". Insbesondere die Gruppendaten werden oben im Speicher festgehalten. In kleinen Speichersystemen (16K oder 20K) können diese Angaben auf Seite 1 der HR-Graphik stehen. Setzen Sie HIMEM: 8192, um Ihr Programm und die HR-Graphik-Seite 1 zu schützen.

HGR 2 imm & def

HGR2

Keine Parameter. Dieses Kommando stellt den HR-Graphik-Betrieb über den ganzen Bildschirm (Anzeige: 280 x 192) ein. Der Bildschirminhalt wird gelöscht und eine schwarze Bildfläche eingestellt. Seite 2 des Speichers (16K - 24K) wird angezeigt. Der Speicher für Textbildschirm wird nicht beeinflußt. Diese Seite des Speichers (und deshalb auch das HGR2-Kommando) steht Ihnen nicht zur Verfügung, wenn Ihr System weniger als 24K-Speicherkapazität aufweist. Bei den Systemen, die darüber verfügen, vergrößert der HGR2-Befehl an Stelle des HGR-Befehls den für Programme verfügbaren Speicherraum auf ein Maximum.

Bei 24K-Systemen müssen Sie HIMEM: 16384 setzen, um die Seite 2 der HR-Graphik vor Ihrem Programm zu schützen (insbesondere die Gruppen, die ganz oben im Speicher festgehalten werden).

ACHTUNG!

Sollte das Tabuwort HGR2 als Anfang eines Variablennamens verwendet werden, dann kann die HGR2-Anweisung ausgeführt werden, ehe die Fehlermeldung

? SYNTAX ERROR

angezeigt wird. Bei der Ausführung einer Anweisung wie bspw.

140 IF X > 150 THEN HGR2PIECES = 12

bleibt der Bildschirm plötzlich frei, und möglicherweise sind die oberen Teile des Programms gelöscht.

Der Befehl

POKE -16301, 0

bewirkt das Umschalten der Nur-Graphik-Anzeige zur Graphik/Text-Parallelanzeige. Wenn er nach HGR2 erteilt wird, werden jedoch die vier Textzeilen von Textseite 2, die nicht ohne weiteres für den Nutzer zugänglich ist, genommen.

HCOLOR imm & def

HCOLOR = aexpr

Mit diesem Befehl wird die HR-Graphik-Farbe eingestellt, die durch den Wert von HCOLOR, der im Bereich von 0 bis 7 liegt, bestimmt wird. Die Bezeichnung der Farben und ihre entsprechenden Zahlenwerte befinden sich in der folgenden Übersicht:

0 - schwarz 1
1 - grün (je nach Bildschirm)
2 - blau (je nach Bildschirm)
3 - weiß 1
4 - schwarz 2
5 - (je nach Bildschirm)
6 - (je nach Bildschirm)
7 - weiß 2

ACHTUNG!

Ein mit HCOLOR=3 (weiß) dazustellender Punkt in HR-Graphik, wird blau, wenn die x-Koordinate des Punktes geradzahlig ist, grün, wenn die x-Koordinate ungeradzahlig ist, und weiß nur dann, wenn sowohl (x,y) als auch (x+1,y) abgebildet werden. Das läßt sich auf die Arbeitsweise der Haushaltsfernsehgeräte zurückführen.

HCOLOR wird nicht durch HGR, HGR2 oder RUN verändert. Bis die erste HCOLOR-Anweisung ausgeführt wird, bleibt die Farbe, mit der die HR-Graphik geschrieben wird, unbestimmt.

Wenn der Befehl während der LR-Graphik-Betriebsweise verwendet wird, hat er keinen Einfluß auf die Farbe, die auf der Anzeige sichtbar wird.

HPLOT imm & def

HPLOT aexpr1, aexpr2

HPLOT TO aexpr3, aexpr4

HPLOT aexpr1, aexpr2 TO aexpr3, aexpr4 [{TO aexpr, aexpr}]

HPLOT mit der ersten Option läßt einen HR-Punkt abbilden, dessen X-Koordinate \aexpr1\ und dessen Y-Koordinate \aexpr2\ heißt. Die Farbe des Punktes wird durch das zuletzt erfüllte HCOLOR vorgegeben. Der Wert von HCOLOR bleibt unbestimmt, wenn er nicht vorher festgelegt wurde.

Die zweite Option realisiert das Ziehen einer Linie vom zuletzt abgebildeten Punkt zu dem Punkt mit den Koordinaten (\aexpr3\ , \aexpr4\ ). Die Farbe der Linie wird von derjenigen bestimmt, in der der zuletzt gedruckte Punkt abgebildet wurde, selbst wenn der Wert von HCOLOR sich zwischenzeitlich verändert haben sollte. Wenn kein Punkt vorher abgebildet wurde, wird auch keine Linie gezogen.
Wenn man HPLOT mit der dritten Option verwendet, wird eine Linie von Punkt (\aexpr1\ , \aexpr2\) auf Punkt (\aexpr3\ , \aexpr4\) gezogen, wobei die Farbe benutzt wird, die durch das letzte HCOLOR-Kommando bestimmt wurde. Die abgebildete Linie kann in der gleichen Anweisung fast unendlich verlängert werden (je nach den Bildschirmgrenzen und der Begrenzung der Anweisung auf 239 Zeichen), indem man die Anweisung durch

TO aexpr5, aexpr6 TO aexpr7, aexpr8

usw. erweitert. Die einzelne Anweisung

HPLOT Ø, Ø TO 279,Ø TO 279,159 TO Ø, 159 TO Ø,Ø

reicht aus, um eine rechteckige Begrenzung um alle vier Seiten des HR-Bildschirms abzubilden.

VERBOTEN!

HPLOT muß ein HGR oder ein HGR2 vorausgehen, um zu vermeiden, daß große Teile des Speichers, einschließlich Ihres Programms und Ihrer Variablen, ruiniert werden.

\aexpr1\ und \aexpr3\ müssen im Bereich von 0 bis 279 liegen.
\aexpr2\ und \aexpr4\ müssen im Bereich von 0 bis 191 liegen.

\ aexpr1\ kann größer als, gleich, oder kleiner als\aexpr3\sein.

\aexpr2\kann größer, gleich oder kleiner als \ aexpr4\sein.

Wenn man versucht, einen Punkt abzubilden, dessen Koordinaten diese Grenzen über-schreiten, wird die Fehlermeldung:

? ILLEGAL QUANTITY ERROR

ausgelöst. Wenn der Bildschirm auf HR-Graphik mit 4 Zeilen Text eingestellt ist, dann werden die Versuche, Punkte abzubilden, deren Y-Koordinaten im Bereich 160 bis 191 liegen, keine sichtbaren Effekte erzielen.

PDL imm & def
PDL (aexpr)

Diese Funktion gibt den gegenwärtigen Wert des Spielreglers (oder PaDdLe), der sich im Bereich von 0 bis 255 bewegt, wieder, wie er durch \aexpr\ festgelegt ist, falls sich \aexpr\ im Bereich von 0 bis 3 befindet. Der Spielregler ist ein variabler Widerstand von 0 bis 150 K-Ohm.

Wenn zwei Spielregler in aufeinanderfolgenden PDL-Anweisungen gelesen werden, kann das Lesen des zweiten vom Lesen des ersten Spielreglers beeinflußt werden. Um genaueres Lesen zu erreichen, lassen Sie immer genügend Platz (mehrere Programmzeilen) zwischen den PDL-Anweisungen oder geben Sie eine kurze Verzögerungsschleife (FOR I=1 TO 1Ø : NEXT I) zwischen die beiden Anweisungen.
Wenn \ aexpr\ negativ oder größer als 255 ist, dann erscheint die Fehlermeldung

? ILLEGAL QUANTITYYERROR.

ACHTUNG!

Wenn \ aexpr\ im Bereich von 4 bis 255 sein sollte, dann gibt die PDL-Funktion eine ziemlich schwer voraussehbare Zahl zwischen 0 und 255 wieder und kann verschiedene Nebeneffekte hervorrufen, von denen einige die Programmausführung stören.
Beispielsweise wird die Verwendung der PDL-Funktion häufig und ziemlich wahllos von einem

"Klickgeräusch" aus dem Lautsprecher des Computers begleitet, wenn \ aexpr\ im Bereich von 204 und 219 liegt.

ACHTUNG!

Wenn N im Bereich von 236 bis 239 liegt, dann kann PDL (N) zu einem
POKE -16540+N, 0

führen, so daß PDL (236) den Graphik-Betrieb, PDL (237) den Textbetrieb usw. (siehe Anlage J) einstellen kann.

Neben der Möglichkeit, mit Hilfe von PDL die Einstellung von vier veränderlichen Spielreglern lesen zu können, besteht für APPLE-SOFT auch noch diejenige, den Zustand von drei Spielknöpfen (Ein/Aus-Schalter) mit Hilfe von verschiedenen PEEK-Kommandos zu lesen. Es kann ferner vier Spielauslöser (TTL-Schalter) an- und abschalten, indem verschiedene POKE-Kommandos benutzt werden (siehe Analge J).

KAPITEL 9

Formative für Darstellungen im HR-Betrieb

Methoden zur Anfertigung von Formativen

In APPLESOFT gibt es fünf spezielle Befehle, die Ihnen gestatten, Formative in HR-Graphiken zu manipulieren, nämlich DRAW, XDRAW, ROT, SCALE und SHLOAD. Bevor diese Befehle verwendet werden können, muß ein Formativ durch eine "Formativdefinition" festgelegt werden. Diese Formativdefinition besteht aus einer Folge von Abbildungsvektoren die in einer Reihe von Bitgruppen im Speicher des APPLE II festgehalten werden. Eine oder mehrere dieser Formativdefinitionen mit ihren Indizes ergeben eine "Formativtabelle", die man über das Tastenfeld erzeugen und auf Platte oder Kassettentonband für zukünftigen Gebrauch sichern kann.

Jede Bitgruppe in einer Formativdefinition ist in drei Abschnitte unterteilt, und jeder Abschnitt kann einen Abbildungsvektor benennen, der angibt, ob ein Punkt abgebildet wird oder nicht und in welche Richtung er sich bewegen soll (hoch, herunter, links oder rechts). DRAW und XDRAW bewirken, daß jede Bitgruppe der Formativdefinition Abschnitt für Abschnitt durchgegangen wird, wobei bei der ersten Bitgruppe begonnen wird. Wird eine Bitgruppe erreicht, in der nur Nullen enthalten sind, dann ist die Formativdefinition abgeschlossen.

So sind die drei Abschnitte (A, B und C) innerhalb einer Bitgruppe angeordnet, die insgesamt die Formativdefinition ausmachen:

Abschnitt	C		B			A		
Bitzahl	7	6	5	4	3	2	1	Ø
benennt:	D	D	P	D	D	P	D	D

Jedes Bitpaar DD bestimmt eine Bewegungsrichtung, und jedes Bit P bestimmt, ob ein Punkt abgebildet wird oder nicht, bevor die Bewegung realisiert wird. Das sieht folgendermaßen aus:

Wenn DD = ØØ	dann: hoch		
Wenn DD = Ø1	dann: rechts	Wenn P = Ø	dann: nein
Wenn DD = 1Ø	dann: runter	Wenn P = 1	dann: ja
Wenn DD = 11	dann: links		

Beachten Sie, daß im letzten Abschnitt, C (die wichtigsten beiden Bit), kein P-Feld

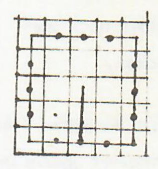
(bei verabsäumter Festlegung: P = Ø) enthalten ist. Demzufolge kann Abschnitt C immer nur eine Bewegung ohne Abbildung eines Punktes benennen.
Jede Bitgruppe kann bis zu drei Abbildungsvektoren darstellen, einen in Abschnitt A, einen in Abschnitt B und einen dritten (nur eine Bewegung) in Abschnitt C.

DRAW und XDRAW bearbeiten die Abschnitte von rechts nach links (d.h. vom wenig bedeutsamen bis zum wichtigsten Bit, also Abschnitt A, dann B und dann C). Für alle Abschnitte der Bitgruppe gilt:

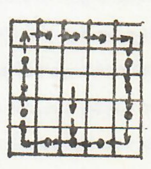
WENN ALLE VERBLEIBENDEN ABSCHNITTE DER BITGRUPPE NUR NULLEN ENTHALTEN, DANN WERDEN DIESE ABSCHNITTE IGNORIERT.
So kann also die Bitgruppe nicht mit einer ØØ-Bewegung (aufwärts, ohne Punktabbildung) in Abschnitt C enden, weil dieser Abschnitt, da er nur Nullen enthält, nicht berücksichtigt wird. In ähnlicher Weise verhält es sich auch, wenn Abschnitt C ØØ lautet und Abschnitt B ØØØ beinhaltet. Dann würde auch Abschnitt B keine Berücksichtigung finden. Und schließlich würde eine ØØØ-Bewegung in Abschnitt A die Formativdefinition abschließen, wenn es nicht noch irgendwo in Abschnitt B oder C ein 1-Bit gäbe.

Nehmen wir an, Sie wollten folgendes Formativ zeichnen:

Zeichnen Sie es erst auf Milimeterpapier einen Punkt pro Quadrat. Dann entscheiden Sie, wo die Zeichnung des Formativs begonnen werden soll. Beginnen wir im Zentrum. Dann geben Sie eine Spur durch jeden Punkt des Formativs vor, wobei Richtungsänderungen nur im Winkel von 90° erfolgen dürfen.



Als nächstes zeichnen Sie das Formativ noch einmal als eine Reihe von Abbildungsvektoren, wobei jeder immer nur eine Stelle vorrückt. Geben Sie den Vektoren, die einen Punkt abbilden, bevor sie vorrücken, ein anderes Aussehen (ein Punkt kennzeichnet den



punktabbildenden Vektor).

"Rollen" Sie jetzt die Vektoren "aus" und schreiben Sie diese in eine Zeile:

↓ ↓ ⟵ ⟵ ↑ ↕ ↕ ↕ ⟶ ⟶ ⟶ ⟶ ↓ ↕ ↕ ↕ ⟵ ⟵

Stellen Sie danch eine Tabelle nach dem folgenden Muster auf:

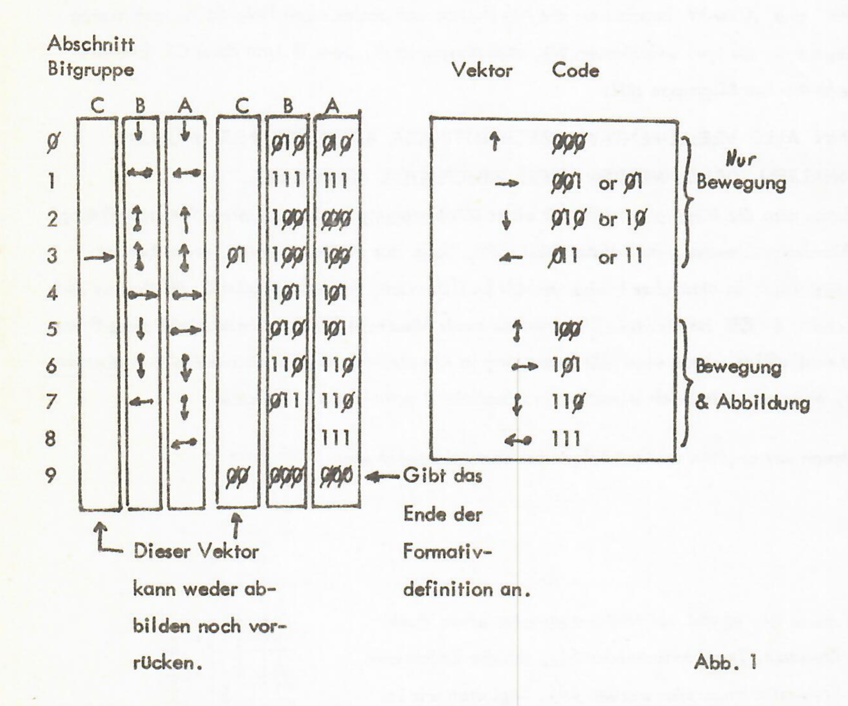


Abb. 1

Bestimmen Sie für jeden Vektor auf der Zeile den Bit-Code und tragen Sie ihn in die nächste verfügbare Spalte in der Tabelle ein. Wenn ein Code nicht paßt, (z.B. kann der Vektor in Abschnitt C keinen Punkt abbilden) oder ØØ (bzw. ØØØ) am Ende der Bitgruppe ist, dann überspringen Sie die Spalte und tragen ihn in der nächsten Spalte ein. Wenn Sie mit der Codierung der Vektoren fertig sind, überprüfen Sie Ihre Arbeit auf Richtigkeit. Stellen Sie danach eine weitere Tabelle nach dem folgenden Muster auf, und übertragen die Vektoren-Codes aus der ersten Tabelle. Codieren Sie jetzt die Vektoreninformationen in eine Reihe von hexadezimalen Bitgruppen, wobei Sie Hexadezimal-Codes aus der

danebenstehenden Aufstellung verwenden:

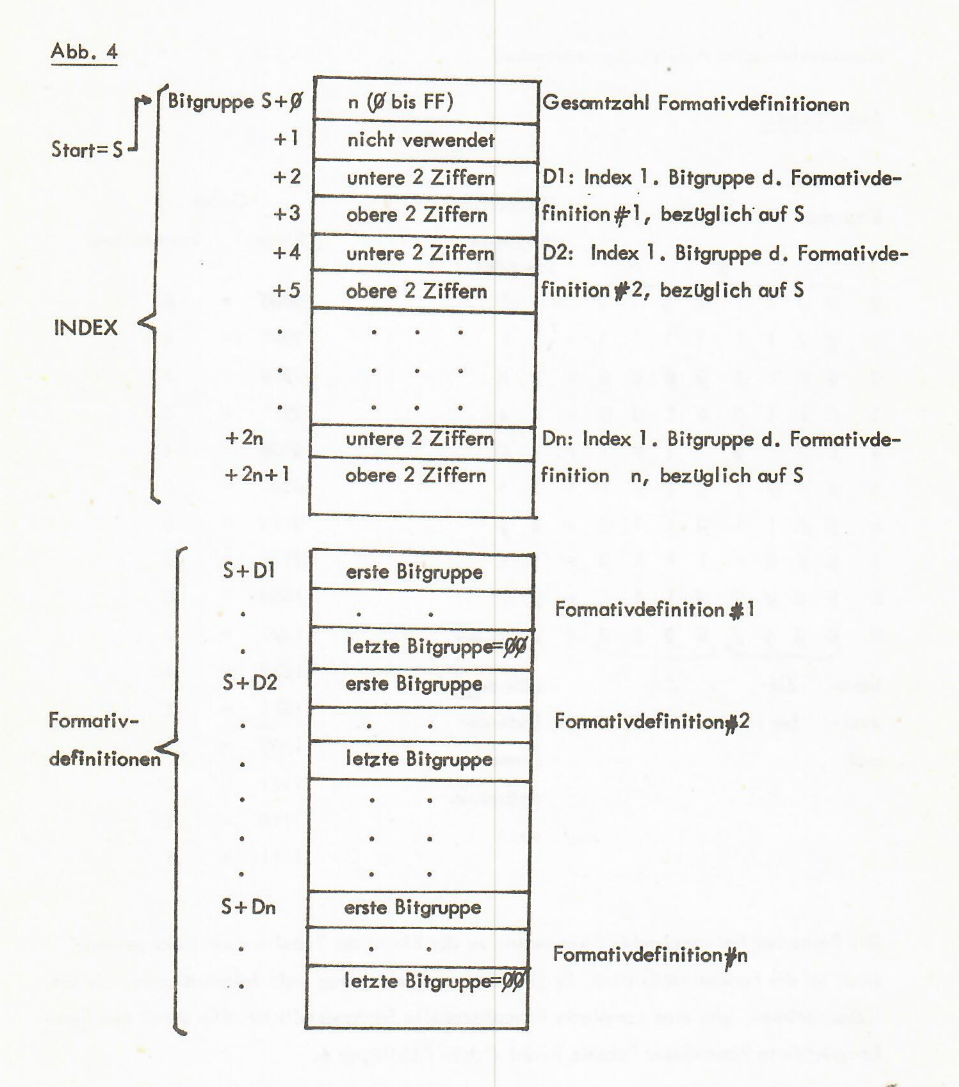
Abb. 2 und 3

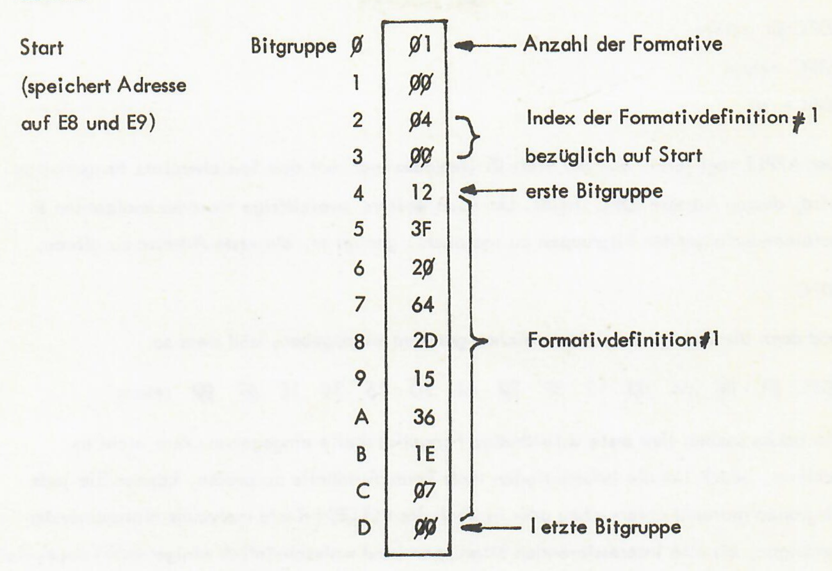
Abschnitt Bitgruppe	C		B			A				Umcodierte Bitgruppe (hexadez.)
Ø	Ø	Ø	Ø	1	Ø	Ø	1	Ø	=	1 2
1	Ø	Ø	1	1	1	1	1	1	=	3 F
2	Ø	Ø	1	Ø	Ø	Ø	Ø	Ø	=	2 Ø
3	Ø	1	1	Ø	Ø	1	Ø	Ø	=	6 4
4	Ø	Ø	1	Ø	1	1	Ø	1	=	2 D
5	Ø	Ø	Ø	1	Ø	1	Ø	1	=	1 5
6	Ø	Ø	1	1	Ø	1	1	Ø	=	3 6
7	Ø	Ø	Ø	1	1	1	1	Ø	=	1 E
8	Ø	Ø	Ø	Ø	Ø	1	1	1	=	Ø 7
9	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	=	Ø Ø ← gibt das Ende der Formativdefinition an.
hexadezimal	Ziffer 1				Ziffer 2					

Codes binär		hexadezimal
ØØØØ	=	Ø
ØØØ1	=	1
ØØ1Ø	=	2
ØØ11	=	3
Ø1ØØ	=	4
Ø1Ø1	=	5
Ø11Ø	=	6
Ø111	=	7
1ØØØ	=	8
1ØØ1	=	9
1Ø1Ø	=	A
1Ø11	=	B
11ØØ	=	C
11Ø1	=	D
111Ø	=	E
1111	=	F

Die Reihe der hexadezimalen Bitgruppen, zu der Sie in der Tabelle oben links gelangt sind, ist die Formativdefinition. Es gibt aber noch ein wenig mehr Informationen, die Sie liefern müssen, ehe eine komplette Formativtabelle fertiggestellt ist. Die durch den Index komplettierte Form dieser Tabelle findet sich in Abbildung 4.

Abb. 4


Start= S
Bitgruppe S+Ø
n (Ø bis FF)
Gesamtzahl Formativdefinitionen
+1
nicht verwendet
+2
untere 2 Ziffern
D1: Index 1. Bitgruppe d. Formativdefinition #1, bezüglich auf S
+3
obere 2 Ziffern
+4
untere 2 Ziffern
D2: Index 1. Bitgruppe d. Formativdefinition #2, bezüglich auf S
+5
obere 2 Ziffern
INDEX
+2n
untere 2 Ziffern
Dn: Index 1. Bitgruppe d. Formativdefinition n, bezüglich auf S
+2n+1
obere 2 Ziffern
S+D1
erste Bitgruppe
Formativdefinition #1
letzte Bitgruppe=ØØ
S+D2
erste Bitgruppe
Formativdefinition #2
letzte Bitgruppe
Formativ-definitionen
S+Dn
erste Bitgruppe
Formativdefinition #n
letzte Bitgruppe=ØØ



Nun können Sie die Formativtabelle in den Speicher des APPLE eingeben. Wählen Sie zunächst eine Startadresse, Für unser Beispiel wollen wir die Hexadezimaladresse 1DFC benutzen. (Beachten Sie, daß diese Adresse unter der obersten verfügbaren Speicheradresse Ihres Systems liegen muß, und nicht in einem Bereich, der gelöscht wird, wenn man HGR oder HGR2 verwendet. Der Speicherplatz 1DFC liegt gerade unterhalb der HR-Graphikseite 1, die durch HGR benutzt wird) Betätigen Sie die RESET-Taste, um das Monitor-Programm einzugeben. Dann tasten Sie die Startadresse Ihrer Formativtabelle ein:

1DFC.

Wenn Sie jetzt die RETURN-Taste betätigen, wird Ihnen der APPLE die Adresse und den Inhalt der Adresse zeigen. So überprüft man eine Adresse, um zu sehen, ob die richtige Zahl eingegeben wurde. Wenn Sie stattdessen einen Doppelpunkt (:) und danach eine Hexadezimalzahl aus zwei Ziffern eingeben, wird diese Zahl auf der bestimmten Adresse gespeichert, sobald Sie die RETURN-Taste betätigen. Versuchen Sie einmal folgendes:

1DFC return .

Was sagt der APPLE über den Inhalt der Speicherstelle 1DFC? Nun versuchen Sie das:

1DFC:Ø1 return

1DFC return

1DFC- Ø1

Der APPLE sagt jetzt, daß der Wert Ø1 (hexadezimal) auf dem Speicherplatz festgehalten wird, dessen Adresse 1DFC heißt. Um noch weitere zweiziffrige Hexadezimalzahlen in aufeinanderfolgenden Bitgruppen zu speichern, genügt es, die erste Adresse zu öffnen,

1DFC: ,

und dann die Zahlen durch Leerzeichen getrennt einzugeben, und zwar so:

1DFC: Ø1 ØØ Ø4 ØØ 12 3F 2Ø 64 2D 15 36 1E Ø7 ØØ return

Sie haben soeben Ihre erste vollständige Formativtabelle eingegeben. Gar nicht so schlimm, oder? Um die Informationen Ihrer Formativtabelle zu prüfen, können Sie jede Bitgruppe getrennt untersuchen oder einfach die RETURN-Taste mehrmals hintereinander betätigen, bis alle interessierenden Bitgruppen (und wahrscheinlich einige mehr) angezeigt wurden. Das läuft etwa so:

```
 1DFC return
1DFC- Ø1
* return
 ØØ Ø4 ØØ
1EØØ- 12 3F 2Ø 64 2D 15 36 1E
* return
1EØ8- Ø7 ØØ DF 1E 23 ØØ ØØ FF
```

Wenn Ihre Formativtabelle richtig aussieht, dann bleibt nur noch, die Startadresse der Formativtabelle zu speichern, wo APPLESOFT sie finden kann (das geschieht automatisch, wenn Sie SHLOAD benutzen, um eine Tabelle von einem Kassettentonband abzurufen). APPLESOFT sucht die vier Hexadezimalziffern der Startadresse der Tabelle auf den hexadezimalen Speicherplatzen E8 (untere 2 Ziffern) und E9 (obere 2 Ziffern). Für die Startadresse der Tabelle in unserem Beispiel (1D FC) würde das ausreichen:

E8:FC 1D return .

Um die Formativtabelle vor dem zufälligen Löschen zu bewahren, das durch Ihr APPLESOFT Programm eintreten könnte, wäre es ein guter Gedanke, HIMEM: (auf hexadezimalen Speicherplätzen 73 und 74) auf die Startadresse der Tabelle zu setzen, nämlich so:

73:FC 1D

Das Gleiche geschieht automatisch, wenn Sie mit Hilfe von SHLOAD die Tabelle vom Kassettentonband einspielen.

Sicherung der Formativtabellen

Damit Sie Ihre Formativtabelle auf Band sichern können, müssen Sie drei Dinge wissen:

1) die Startadresse der Tabelle (in unserem Beispiel 1DFC)

2) die letzte Adresse der Tabelle (in unserem Beispiel 1EØ9)

3) die Differenz zwischen 1) und 2) (in unserem Beispiel ØØØD)

Punkt 3, die Differenz zwischen der letzten und der ersten Adresse der Tabelle muß in den hexadezimalen Speicherplätzen Ø (untere 2 Ziffern) und 1 (obere 2 Ziffern) festgehalten werden:

Ø:ØD ØØ return

Jetzt können Sie erst die Länge der Tabelle, die auf Platz Ø bis 1 gespeichert wird, und dann die Formativtabelle selbst, die auf den Plätzen Startadresse bis letzte Adresse gespeichert wird, "schreiben" (d.h. auf Kassettentonband aufnehmen):

Ø:1W IDFC.1EØ9W

Betätigen Sie erst die RETURN-Taste, wenn in Ihrem Kassettenrecorder eine zurückgespulte Kassette liegt, und das Gerät auf Aufnahme eingestellt ist (Play- und Record-Taste zugleich gedrückt). Betätigen Sie jetzt die RETURN-Taste des Computers.
Um nun die Tabelle verwenden zu können, spulen Sie die Kassette zurück, und lassen Sie sie abspielen (Play-Taste drücken). Dem Computer (APPLESOFT) geben Sie ein:

SHLOAD return

Sie müßten jetzt ein erstes "Piep" hören, wenn die Länge der Tabelle gelesen wurde, und ein zweites "Piep", wenn die Tabelle selbst gelesen wurde.

Benutzung der Formativtabellen

Sie können nun ein APPLESOFT-Programm schreiben und Formativtabellenbefehle DRAW, XDRAW, ROT und SCALE benutzen.

Hier nun ein APPLESOFT-Musterprogramm, welches unser definiertes Formativ ausschreiben und 16° drehen wird, dann das Gleiche wiederholen wird, wobei jede Wiederholung größer ist als die vorherige.

```
1Ø HGR
2Ø HCOLOR = 3
3Ø FOR R = 1  TO 5Ø
4Ø ROT = R
5Ø SCALE = R
6Ø DRAW 1 AT 139,79
7Ø NEXT R
```

Damit ein einzelnes "Quadrat" sichtbar wird, fügen Sie die Zeile ein:

```
65 END
```

Damit es pausiert und dann jedes Quadrat auslöscht, nachdem es gezeichnet wurde, müssen Sie diese Zeilen einfügen:

```
63 FOR I=Ø TO 1ØØØ: NEXT I
65 DRAW 1 AT 139, 79
```

DRAW imm & def
DRAW aexpr1 AT aexpr2, aexpr3
DRAW aexpr1

DRAW in der ersten Variante läßt ein Formativ in HR-Graphik entstehen, das in dem Punkt beginnt, dessen X-Koordinate \aexpr2 \ ist und dessen Y-Koordinate \aexpr3\ ist. Das gezeichnete Formativ ist die\aexpr1 \ -te Formativdefinition in der Formativtabelle, die vorher mit Hilfe des SHLOAD-Befehls geladen wurde. (Eine Formativtabelle kann auch in den Speicher des APPLE im Hexadezimal-Code eingegeben werden, indem

man das Monitorprogramm verwendet.)

\ aexpr1 \ muß im Bereich von Ø bis n liegen, wobei n die Zahl der Formativdefinitionen (von Ø bis 255) ist, die in der Bitgruppe Ø der Formativtabelle angegeben wird. \ aexpr2\ muß im Bereich von Ø bis 278 liegen, und\ aexpr3\ im Bereich von Ø bis 191. Falls einer dieser Bereiche überschritten wird, dann wird die Fehlermeldung

? ILLEGAL QUANTITY ERROR

angezeigt.

Die Farbe, die Drehung und der Maßstab des zu zeichnenden Formativs müssen bestimmt worden sein, ehe das DRAW-Kommando ausgeführt wird.

Die zweite Variante ist der ersten ähnlich. Der Unterschied besteht darin, daß hier die Zeichnung des Formativs in dem Punkt beginnt, der als letzter durch die Ausführung eines HPLOT-, DRAW-oder XDRAW-Befehls abgebildet wurde.

ACHTUNG!

Wenn der Befehl ausgegeben wird, ohne daß sich eine Formativtabelle im Computer befindet, kann das dazu führen, daß das System "hängt". Um es wieder zu regenerieren, benutzt man **reset ctrl C return**. Es kann unter o.g. Umständen auch willkürliche "Formative" über den gesamten HR-Graphikbereich des Speichers zeichnen, wobei möglicherweise Ihr Programm zerstört werden kann, auch wenn Sie nicht im Graphikbetrieb arbeiten sollten.

XDRAW imm &def
XDRAW aexpr1 [AT aexpr2, aexpr3]

Durch diesen Befehl wird der gleiche Effekt erzielt wie durch DRAW, nur mit dem Unterschied, daß die Farbe, in der das Formativ gezeichnet wird, die Komplementärfarbe derjenigen ist, mit der die bestehenden Punkte abgebildet wurden. Komplementärfarben sind bspw.:

schwarz - weiß
blau - grün

Der Zweck des XDRAW-Befehls besteht darin, auf einfache Weise zu löschen, ohne den Untergrund mit "auszuradieren". Wenn Sie nämlich XDRAW für ein Formativ geben, und dann ein zweites für das gleiche Formativ, dann verschwindet das Formativ von dem Bild, auf dem das Formativ gezeichnet wurde, ohne das Bild zu löschen.

ACHTUNG!

Siehe Vorsichtsmaßregeln unter DRAW.

ROT imm & def

ROT = aexpr

Mit diesem Befehl wird eine Drehung des durch DRAW oder XDRAW gezeichneten Formativs um einen bestimmten Winkel realisiert. Der Rotationsbetrag wird durch\aexpr\ festgelegt, der sich im Bereich von Ø bis 255 befinden muß.

ROT = Ø läßt das Formativ so zeichnen, wie es bestimmt wurde. Mit ROT = 16 erreicht man, daß das Formativ um 90° im Uhrzeigersinn gedreht abgebildet wird, mit ROT = 32 um 180° usw. Der ganze Prozeß wiederholt sich ab ROT = 64 von neuem. Bei SCALE = 1 werden nur vier Rotationswerte anerkannt (Ø, 16, 32, 48), bei SCALE = 2 acht usw. Unerkannte Rotationswerte werden verursachen, daß das Formativ mit der Orientierung durch die (gewöhnlich) nächstkleinere Rotation abgebildet wird.

ROT wird nur als Tabuwort erkannt, wenn das erste, vom Leerzeichen verschiedene Zeichen das Gleichheitszeichen (=) ist.

SCALE imm & def

SCALE = aexpr

Mit dieser Anweisung wird die Maßstabsgröße des durch DRAW oder XDRAW zu zeichnenden Formativs festgelegt. Sie reicht von 1 (Punkt-für-Punkt-Reproduktion der Formativdefinition) bis 255 (jeder Vektor 255 mal erweitert). \ aexpr\ gibt den Maßstab an. Beachten Sie: SCALE=Ø bedeutet Maximalgröße und nicht einzelner Punkt.
SCALE wird nur dann als Tabuwort identifiziert, wenn das nächste, vom Leerzeichen verschiedene Zeichen das Gleichheitszeichen (=) ist.

SHLOAD imm & def

SHLOAD

Hiermit wird eine Formativtabelle von dem Kassettentonband geladen. Die Formativtabelle wird gleich unterhalb von HIMEM: geladen, und HIMEM: wird gleich unter die Formativtabelle gesetzt, um sie zu schützen. Die Startadresse der Formativtabelle wird den Formativzeichenprogrammen von APPLESOFT automatisch gegeben. Wenn eine zweite, die erste ersetzende Formativtabelle geladen wird, sollte man HIMEM: neu setzen, um Vergeudung von Speicherraum zu vermeiden. Die Formativtabellenbänder werden hergestellt, indem man die Anweisungen zu Anfang des Kapitels benutzt.
Bei Systemen mit 16K-Speichern löscht HGR den über 8K liegenden Speicherinhalt, d.h. von Speicherplatz 8192 bis 16383. Damit die Formativtabelle durch den SHLOAD-Befehl wirklich unter die HR-Graphikseite 1 geladen wird, setzen Sie HIMEM:8192, bevor Sie den SHLOAD-Befehl ausführen lassen. Bei Systemen mit 24K-Speichern verwenden Sie bitte nicht HGR2 (was den Speicherinhalt von Platz 16384 bis 24575 löscht) oder setzen Sie sonst HIMEM:16384 vor dem SHLOAD und benutzen dann nicht HGR. Wenn Sie sich sicher sind, daß noch genügend freie Speicherkapazität oberhalb der Stelle 24575 vorhanden ist, wo Ihre Formativtabelle aufgenommen werden kann, dann brauchen Sie sich um nichts Sorgen zu machen.

Der SHLOAD-Befehl kann nur durch reset unterbrochen werden. Leitet das Tabuwort SHLOAD einen Variablennamen ein, dann kann der aus dem SHLOAD folgende Befehl ausgeführt werden, ehe die Fehlermeldung

? SYNTAX ERROR

angezeigt wird. Die Anweisung

SHLOADER = 5

läßt das System "hängen", während APPLESOFT unendlich lange auf ein über Kassettenrecorder eingespieltes Programm wartet. Geben Sie reset ctrl C ein, um den Computer wieder in Gang zu bekommen.

KAPITEL 10

Einige mathematische Funktionen

Die eingebauten Funktionen

Alle Funktionen können benutzt werden, wo immer ein Ausdruck des gleichen Typs verwendet wird. Sie können in der Sofort- und der verzögerten Ausführung eingesetzt werden. Nachfolgend nun eine kurze Beschreibung einiger arithmetischer Funktionen von APPLESOFT. Andere Funktionen werden in den sich mit ähnlichen Anweisungen befassenden Abschnitten beschrieben.

SIN (aexpr)

gibt den Sinus von \aexpr\ im Bogenmaß an.

COS (aexpr)

gibt den Cosinus von \aexpr\ im Bogenmaß an.

TAN (aexpr)

gibt den Tangens von \aexpr\ im Bogenmaß an.

ATN (aexpr)

gibt den Arcustangens von \aexpr\ im Bogenmaß an. Der wiedergegebene Winkel liegt im Bereich $-\frac{\pi}{2}$ bis $+\frac{\pi}{2}$.

INT (aexpr)

gibt die größte ganze Zahl wieder, die < \aexpr\ oder = \aexpr\ ist.

RND (aexpr)

gibt eine wahlfreie reelle Zahl wieder, die entweder größer als bzw. gleich Ø ist, aber kleiner als 1.

Wenn \aexpr\ größer als Null, dann erzeugt RND (aexpr) bei jeder Verwendung eine neue wahlfreie Zahl.

Wenn \aexpr\ kleiner als Null, dann erzeugt RND (aexpr) immer die gleiche wahlfreie Zahl, wenn es mit dem gleichen aexpr verwendet wird, als ob eine ständige Tabelle der wahlfreien Zahl in den APPLE eingebaut wäre. Wenn ein besonders negatives Argument verwendet wird, um eine wahlfreie Zahl zu erzeugen, dann werden aufeinanderfolgende wahlfreie Zahlen, die mit positivem Argument erzeugt wurden, in der gleichen Reihenfolge stehen. Der Hauptgrund für die Verwendung eines negativen Arguments für RND besteht darin, eine wiederholbare Reihe von wahlfreien Zahlen zu initiieren (oder zu "pflanzen").

Das erweist sich als außerordentlich nützlich in Fehlersuchprogrammen, in denen RND zur Anwendung kommt. Wenn \aexpr\ = Ø, dann gibt RND (aexpr) die letzte erzeugte wahlfreie Zahl wieder (CLEAR und NEW beeinträchtigen das nicht). Manchmal ist das leichter als die Zuordnung einer Variablen zu dieser letzten wahlfreien Zahl, um diese zu sichern.

SGN (aexpr)

gibt -1 wieder, wenn \aexpr\<Ø, gibt Ø wieder, wenn \aexpr\=Ø und gibt 1 wieder, wenn\aexpr\>Ø.

ABS (aexpr)

gibt den absoluten Wert eines \aexpr\ wieder, d.h. \aexpr\, wenn\aexpr\>=Ø und - \aexpr\ wenn \aexpr\<Ø.

SQR (aexpr)

gibt die positive Quadratwurzel wieder. Diese ist eine spezielle Ausführung, die schneller ist als ^.5 (= dt. $0,5^2$)

EXP (aexpr)

erhebt e(bis zu 6 Stellen nach dem Komma, d.h. e=2,718289) in die durch \aexpr\ bestimmte Potenz.

LOG (aexpr)

gibt den natürlichen Logarithmus von\aexpr\ wieder.

Die abgeleiteten Funktionen

Die nachfolgend aufgeführten Funktionen können, obwohl nicht Bestandteil von APPLESOFT-BASIC, doch mit Hilfe der bestehenden BASIC-Funktionen berechnet werden und leicht unter Verwendung von DEF FN ausgeführt werden.

Secans:

SEC(X) = 1/COS(X)

Cosecans:

CSC(X) = 1/SIN(X)

Cotangens:

COT(X) = 1/TAN(X)

Arcussinus:

ARCSIN(X) = ATN(X/SQR(-X *X+1))

Arcuscosinus:

ARCCOS(X) = -ATN(X/SQR(-X*X+1))+1.5708

Arcussecans:

ARCSEC(X) = ATN(SQR(X*X-1))+(SGN(X)-1)* 1.5708

Arcuscosecans:

ARCCSC(X) = ATN(1/SQR(X*X-1))+(SGN(X)-1)* 1.5708

Arcuscotangens:

ARCCOT(X) = -ATN(X)+1.5708

Hyperbelsinus:

SINH(X) = (EXP(X)-EXP(-X))/2

Hyperbelcosinus:

COSH(X) = (EXP(X)+EXP(-X))/2

Hyperbeltangens:

TANH(X) = -EXP(-X)/(EXP(X)+EXP(-X))*2+1

Hyperbelsecans:

SECH(X) = 2/(EXP(X)+EXP(-X))

Hyperbelcosecans:

CSCH(X) = 2/(EXP(X)-EXP(-X))

Hyperbelcotangens:

COTH(X) = EXP(-X)/(EXP(X)-EXP(-x))* 2+1

Hypberbelarcussinus:

ARGSINH(X) = LOG(X+SQR(X*X+1))

Hyperbelarcuscosinus:

ARGCOSH(X) = LOG(X+SQR(X*X-1))

Hyperbelarcustangens:

```
ARGTANH(X) = LOG((1+X)/(1-X))/2
```

Hyperbelarcussecans:

```
ARGSECH(X) = LOG((SQR(-X*X+1)+1)/X
```

Hyperbelarcuscosecans:

```
ARGCSCH(X) = LOG(SGN(X)*SQR(X*X+1)+1)/X
```

Hyperbelarcuscotangens:

```
ARGCOTH(X) = LOG((X+1)/(X-1))/2
```

A MOD B:

```
MOD(A) = INT((A/B-INT(A/B))*B+.Ø5)*SGN(A/B)
```

ANLAGEN

Anhang A

Laden und Starten des Programms APPLESOFT BASIC

APPLE Computer Inc. bietet zwei Versionen der BASIC-Programmiersprache an. Ganzzahlen-BASIC, das in dem Handbuch APPLE II BASIC Programming Manual beschrieben wird, ist eine sehr schnelle BASIC-Version, die für viele Anwendungen geeignet ist, insbesondere für Bildungszwecke, Computerspiele und grafische Darstellungen. Die zweite BASIC-Version wird "APPLESOFT" genannt und ist besser für die meisten geschäftlichen und wissenschaftlichen Anwendungsbereiche geeignet.

APPLESOFT BASIC ist in zwei Versionen verfügbar. Die Firmware APPLESOFT wird mit APPLESOFT in einem ROM auf einer Leiterkarte geliefert, die direkt in den APPLE II eingesteckt wird. In dieser Ausführung ist das Betätigen eines Schalters und das Drücken zweier Tasten notwendig, um den APPLE II mit APPLESOFT arbeiten zu lassen. Neben diesem Komfort spart die Tatsache, daß APPLESOFT in einem ROM verfügbar ist, einen Speicherraum von etwa 1Ø KByte und außerdem viel Zeit, die benötigt würde, um die Sprache bei jeder Benutzung von einem Kassettenband zu laden. Der Hauptteil dieses Handbuches setzt voraus, daß Sie die Firmwarekarte APPLESOFT besitzen. Falls Sie aber die Bandkassettenversion von APPLESOFT verwenden, dann orientieren Sie sich bitte im Teil II dieses Anhanges über gesonderte Anweisungen und Hinweise über Unterschiede Ihrer APPLESOFT-Version zu derjenigen, die im Rest dieses Handbuches beschrieben wird.

Zur Beachtung: In diesem Handbuch bedeutet das Wort "reset", daß die Taste mit der Bezeichnung RESET zu drücken ist; das Wort "return" besagt, daß die Taste RETURN zu betätigen ist, und "ctrl B" bedeutet, daß die Taste B während des Niederhaltens der Taste CTRL gedrückt wird.

Wichtiger Hinweis:

Neben der Funktion, Sie zur Eingabe in den Computer zu veranlassen, hat das Kennungszeichen die Aufgabe, auf einen Blick erkennbar zu machen, auf welche Sprache zu reagieren der Computer zur Zeit programmiert ist. Bisher haben Sie z. B. folgende beiden Kennungszeichen gesehen:

✱ für das Monitorprogramm (wenn Sie RESET betätigen)

> für das APPLE Ganzzahlen-BASIC (das normale Ganzzahlen-BASIC)

Nun führen wir ein drittes Kennungszeichen ein:

] für APPLESOFT Gleitkomma-BASIC.

Ein Blick auf dieses Kennungszeichen sagt Ihnen sofort, welche Sprache im Computer resident ist.

Teil 1: FIRMWARE APPLESOFT

Einsetzung der APPLESOFT Firmware-Karte

Die APPLESOFT-Firmware-Karte wird einfach in eine Aufnahmevorrichtung im APPLE II eingesteckt. Beachten Sie dabei aber folgende Anweisungen sorgfältig:

1. Schalten Sie den APPLE ab: das ist sehr wichtig, um den Computer nicht zu beschädigen.

2. Entfernen Sie den Deckel vom APPLE II. Das geschieht durch Hochziehen des Deckels an der Hinterkante (der Kante, die am weitesten von der Tastatur entfernt ist), bis die beiden Eckenbefestiger beiseite springen. Heben Sie die Hinterkante nicht weiter an, sondern schieben Sie den Deckel rückwärts, bis er freikommt.

3. Innerhalb des APPLE II, hinter der Schaltungsplatte, befindet sich eine Reihe von acht langen schmalen Schlitzen. Der am weitesten links befindliche (von der Tastatur aus gesehen) ist Schlitz Nr. Ø; der am weitesten rechte ist Schlitz Nr. 7. Halten Sie die APPLESOFT-Karte so, daß ihr Schalter zur Rückseite des Computers zeigt; setzen Sie den mit "Fingern" versehenen Teil der Karte in den linken Schlitz Nr. Ø ein. Die "Finger" gleiten mit etwas Reibung in den Schlitz und sitzen dann fest. Die APPLESOFT-Karte muß in Schlitz Nr. Ø gesteckt werden.

4. Der Schalter auf der Rückseite der APPLESOFT-Karte soll teilweise aus dem rückwärtigen Schlitz am APPLE II herausstehen.

5. Wiedereinsetzen des Deckels des APPLE: Schieben Sie zuerst die Vorderkante an ihren Platz und drücken Sie dann die beiden hinteren Ecken nieder, bis sie einrasten.

6. Nun schalten Sie den APPLE II ein.

Anwendung der APPLESOFT-Firmware-Karte

Befindet sich der Schalter der APPLESOFT-Karte in Abwärtsstellung, beginnt der APPLE II mit der Arbeit im Ganzzahlen-BASIC, wenn Sie mit "reset ctrl B" arbeiten (das bedeutet in der Sprache dieses Handbuches: Drücken Sie die Taste RESET, dann halten Sie die Taste CTRL gedrückt, während Sie B betätigen). Jetzt werden Sie das Kennungszeichen > sehen, welches Ganzzahlen-BASIC anzeigt.

Befindet sich der Schalter in Aufwärtsstellung, bringt "reset ctrl B" APPLESOFT BASIC anstelle von Ganzzahlen-BASIC hervor. Das Kennungszeichen] sagt Ihnen, daß Sie in APPLESOFT arbeiten.

Bei Anwendung des Plattensystems (Disk Operating System) wählt der Computer automatisch - je nach Erfordernis - Ganzzahlen-BASIC oder APPLESOFT. Es spielt dabei keine Rolle, auf welche Position der Schalter eingestellt wurde.

Sie können auch von Ganzzahlen-BASIC zur APPLESOFT und umgekehrt wechseln, ohne den Schalter oder die Firmware-Karte zu betätigen. Um den Computer für APPLESOFT einzustellen, gehen Sie folgendermaßen vor:

"reset CØ8Ø return"
"crtl B return"

Um den Computer auf Ganzzahlen-BASIC einzustellen, sind folgende Schritte notwendig:

"reset CØ81 return"
"ctrl B return"

Ein weiterer wichtiger Hinweis:

Wenn Sie versehentlich RESET betätigen und sich im Monitor wiederfinden - wie es durch

das Kennungszeichen * angezeigt wird - können Sie zu APPLESOFT BASIC zurückkehren, wobei APPLESOFT und Ihr Programm unversehrt bleiben, wenn Sie folgende Tasten betätigen:

"ctrl C return".

Teil 2: KASSETTENBAND APPLESOFT

APPLESOFT II BASIC wird mit jedem APPLE II kostenlos als Kassettenband zur Verfügung gestellt. APPLESOFT BASIC nimmt - wenn es vom Kassettenband geladen wird - einen Speicherraum von etwa 1Ø KBytes ein, und deshalb ist ein Computer von 16 KBytes oder noch größerem Speicher nötig, um die Kassettenversion des APPLESOFT BASIC anzuwenden.

Starten des APPLESOFT mit KASSETTENBAND

Verfahren Sie folgendermaßen, um APPLESOFT von Ihrem Kassettengerät zu laden:

1. Starten Sie Ganzzahlen-BASIC durch Betätigen von RESET CTRL B. Falls Sie mit diesem Vorgang nicht vertraut sind, schlagen Sie in Ihrem Handbuch APPLE INTEGER BASIC Programming Manual nach. Sie erkennen, daß Sie in Ganzzahlen-BASIC sind, wenn Sie das Kennungszeichen > auf dem Anzeigenschirm sehen, gefolgt von einem blinkenden Quadrat ("cursor").

2. Legen Sie das APPLESOFT-Band (Teil Nr. A2TØØØ4) in Ihren Kassettenrekorder ein und lassen Sie das Band bis zum Anfang zurücklaufen.

3. Geben Sie "LOAD" ein.

4. Drücken Sie die Abspieltaste des Rekorders und starten Sie das Band.

5. Drücken Sie die Taste mit der Bezeichnung RETURN auf der Tastatur des APPLE II. Dadurch wird der blinkende "cursor" verschwinden. Nach 5 bis 2Ø s gibt der APPLE II ein akustisches Signal, das anzeigt, daß die Informationen vom Band nunmehr in den Computer eingespeist werden. Nach etwa 1 - 1/2 min ertönt ein weiteres Signal und das Kennungszeichen > , gefolgt von dem "cursor" , erscheint wieder.

6. Stoppen Sie den Bandrekorder und lassen Sie das Band zurücklaufen. APPLESOFT ist nunmehr im Computer.

7. Schreiben Sie nun RUN und drücken Sie die RETURN-Taste. Auf dem Display erscheint der Copyright-Vermerk für APPLESOFT II und das APPLESOFT-Kennungszeichen] .

Es kann vorkommen, daß Sie versehentlich die RESET-Taste drücken und ins Monitor-Programm geraten, was durch das Kennungszeichen * angezeigt wird. Sie können zu APPLESOFT zurückkehren, wobei Ihr Programm und APPLESOFT selbst noch unversehrt sein werden, indem Sie tasten:

ØG RETURN.

Führt das nicht zum Erfolg, werden Sie APPLESOFT nochmals vom Kassettenband laden müssen.

Das Tasten von "ctrl C" oder "ctrl B" wird Sie vom Monitorprogramm zu APPLE Ganzzahlen-BASIC bringen; dadurch wird APPLESOFT gelöscht.

In diesem Handbuch bedeutet "reset" das Drücken der RESET-Taste, "return" das Drücken der RETURN-Taste und "ctrl B" das Betätigen von B bei gleichzeitigem Niederhalten der CTRL-Taste.

Unterschiede zwischen APPLESOFT-Firmware und APPLESOFT-Kassette

APPLESOFT auf Kassettenband (Teil Nr. A2TØØØ4) funktioniert nicht genauso wie es die Firmware-Version von APPLESOFT tut, die sich in einem ROM auf einer einsteckbaren Leiterkarte (Teil Nr. A2BØØØ9X) befindet. Der Hauptteil dieses Handbuchs beschreibt die Firmware-Version von APPLESOFT. Die folgenden Erläuterungen behandeln die Unterschiede zwischen der APPLESOFT-Kassette und der APPLESOFT-Firmware.
Da das Kassettenband APPLESOFT annähernd 1Ø KBytes Speicherraum einnimmt (und der Computer weitere 2 KBytes benötigt), kann Kassetten-APPLESOFT nicht bei APPLEs mit weniger als 16 KBytes Speicherraum benutzt werden. Mit geladenem Kassetten-APPLESOFT ist der niedrigste, dem Benutzter verfügbare Speicherplatzadresse etwa 123ØØ. APPLESOFT-Firmware ist nicht in einem RAM-Speicher resident und kann somit (ohne Graphiken mit hoher

Auflösung) bei kleineren Systemen verwendet werden.

VERBOTEN!

HGR ist nicht in der APPLESOFT-Kassette verfügbar. Das HGR-Kommando löscht "Seite 1" des Graphikenspeichers (8K bis 16K) für Graphiken mit hoher Auflösung. Da Kassetten-A..PPLESOFT diesen Abschnitt im Speicher teilweise einnimmt, wird ein Versuch, HGR anzuwenden, APPLESOFT löschen und kann auch Ihre Programme löschen. Das HGR2-Kommando kann sowohl im ROM, als auch bei der Kassettenversion von APPLESOFT verwendet werden, ist aber nur verfügbar, falls Ihr APPLE einen Speicher von wenigstens 24 K enthält. Deshalb bietet Kassetten-APPLESOFT in einem System mit weniger als 24K Speicherkapazität keine Graphiken mit hoher Auflösung.

Das Kommando POKE-163Ø1,Ø verwandelt jede den ganzen Schirm einnehmende Graphik-Darstellungsweise in eine gemischte Graphik-plus-Text-Darstellungsweise. Wird es aber nach HGR2 gegeben, werden die vier Textzeilen von Seite 2 des Textspeichers genommen. In der Kassettenversion von APPLESOFT nimmt APPLESOFT selbst die Seite 2 des Textspeichers ein, so daß eine gemischte Darstellungsweise "Graphik mit hoher Auflösung plus Text" nicht verfügbar ist.

Bei Ganzzahlen-BASIC und bei APPLESOFT auf der Firmware-Karte können Sie nach einem zufälligen oder beabsichtigten Drücken der RESET-Taste zu Ihrem Programm zurückkehren, indem Sie "ctrl C return" anwenden. Um dasselbe bei Kassetten-APPLESOFT zu bewerkstelligen, müssen Sie "ØG return" anwenden (schreiben Sie Ø, dann G und drücken Sie die RETURN-Taste). Wenn Sie Kassetten-APPLESOFT verwenden, stellt "reset ctrl C return" Ganzzahlen-BASIC als Ihre Programmiersprache wieder her, dadurch wird APPLESOFT gelöscht.

Kurz gesagt: überall dort, wo es in diesem Handbuch heißt "reset ctrl C return" anzuwenden, müssen Kassetten-APPLESOFT-Benutzer stattdessen "reset ØG return" anwenden. Soll laut Handbuch "reset ctrl B return" verwendet werden, so können Sie das tun, müssen dann aber APPLESOFT vom Band neu laden.

Bei Kassetten-APPLESOFT verwenden Sie CALL 11246 (anstatt CALL 6245Ø), um den

HGR2-Schirm zu löschen und Schwarz hervorzurufen. Verwenden Sie CALL 1125Ø (anstatt CALL 62454), um den HGR2-Schirm zu löschen und die zuletzt durch HPLOT gegebene HCOLOR hervorzurufen. Erfolgt das, ehe das HGR2-Kommando das erste Mal gegeben wurde, können diese CALLs APPLESOFT löschen.

Anhang B

Redigieren eines Programms

Die meisten normalen Menschen machen gelegentlich Fehler ... besonders, wenn sie Computerprogramme schreiben. Um das Korrigieren solcher Versehen zu erleichtern, hat APPLE eine Anzahl einzigartiger Redigiereinrichtungen in das APPLESOFT BASIC aufgenommen. Um sie richtig einsetzen zu können, müssen Sie sich zuerst vertraut mit den Funktionen von vier speziellen Tasten auf der Tastatur des APPLE II machen. Es sind die "escape"-Taste mit der Bezeichnung ESC, die Wiederholungstaste REPT und die Links- und die Rechtspfeiltaste, die mit einem Links- bzw. einem Rechtspfeil versehen sind.

ESC

Die "escape"-Taste befindet sich ganz links in der zweiten Reihe von oben. Sie wird IMMER mit einer weiteren Taste (wie A, B, C oder D) auf folgende Weise verwendet: Zum Beispiel drücken Sie ESC und lassen Sie sie los, danach drücken Sie A und lassen sie los ... also nacheinander.

Diese Operation oder Folge der ESC- und einer anderen Taste wird "escape A" geschrieben. Für das Redigieren werden vier "escape"-Funktionen verwendet:

"escape A" bewegt den "cursor" nach rechts,
"escape B" bewegt den "cursor" nach links,
"escape C" bewegt den "cursor" nach unten,
"escape D" bewegt den "cursor" nach oben.

Bei Betätigung der "escape"- und der zweiten Taste kann der "cursor" an jede Stelle auf dem Bildschirm bewegt werden, ohne daß dadurch etwas bereits dort Angezeigtes oder etwas im Speicher Befindliches gestört wird.

RECHTSPFEIL-Taste

Die Rechtspfeil-Taste bewegt den "cursor" nach rechts. Es ist die Taste auf der Tastatur, die am meisten Zeit spart, da sie nicht nur den "cursor" bewegt, sondern ALLE ZEICHEN UND SYMBOLE, über die er sich hinweg bewegt. IN DEN SPEICHER DES APPLE II EINKO-

PIERT, genauso als hatten Sie sie selbst mittels der Tastatur eingegeben. Der TV-Display wird nicht verändert, wenn Sie die Rechtspfeil-Taste betätigen.

LINKSPFEIL-Taste

Die Linkspfeil-Taste bewegt den "cursor" nach links. Immer wenn sich der "cursor" nach links bewegt, WIRD EIN ZEICHEN AUS DER PROGRAMMZEILE GELÖSCHT, DIE SIE GERADE SCHREIBEN, ungeachtet, worüber sich der "cursor" gerade hinwegbewegt. Der TV-Display wird nicht verändert, wenn Sie die Linkspfeil-Taste benutzen. Gewöhnlich kann die Linkspfeil-Taste nicht benutzt werden, um den "cursor" in die sich am weitesten links befindliche Spalte zu bewegen: um das zu erreichen, verwenden Sie "escape B".

REPT

Die REPT-Taste wird zusammen mit einer anderen Zeichentaste der Tastatur verwendet. Sie bewirkt, daß ein Zeichen so lange wiederholt wird, wie sowohl die Taste für dieses Zeichen, als auch die REPT-Taste niedergehalten werden. Nunmehr sind Sie in der Lage, diese Redigierfunktionen zu nutzen, um Zeit zu sparen, wenn Sie Veränderungen oder Korrekturen an Ihrem Programm vorzunehmen haben. Es folgenden nun einige Beispiele, wie das erfolgen soll.

Beispiel 1 - Korrigieren von Tippfehlern

Gesetzt den Fall, Sie haben ein Programm durch Schreiben eingegeben, und wenn Sie es arbeiten lassen, druckt der Computer SYNTAX ERR aus und bleibt stehen; dabei zeigt er das Kennungszeichen] und den blinkenden "cursor". Geben Sie folgendes Programm ein und lassen Sie es laufen. Beachten Sie, daß "PRIMT" und "PREGRAM" absichtlich falsch geschrieben wurden. Nachstehend ist annähernd dargestellt, welches Bild sich auf Ihrem TV-Display darstellen wird:

```
]1Ø PRIMT "THIS IS A PREGRAM"
]2Ø GOTO 1Ø
]RUN
 ?SYNTAX ERR IN 1Ø
]▯
```

Nun schreiben Sie das Wort LIST und drücken Sie "return".

```
]LIST
  1Ø PRIMT "THIS IS PREGRAM"
  2Ø GOTO 1Ø
]□
```

Um den "cursor" bis zum Beginn der Zeile 1Ø zu bewegen, schreiben Sie dreimal "escape D" und danach "escape B". Beachten Sie: Es ist notwendig "escape B" zu verwenden, um den "cursor" auf der ersten Stelle in der Zeilenzahl zu positionieren. Nunmehr wird der TV-Schirm folgendes anzeigen:

```
]LIST
 1Ø PRIMT "THIS IS A PREGRAM"
  2Ø GOTO 1Ø
]
```

Nun drücken Sie die Rechtspfeil-Taste sechsmal, um den "cursor" auf den Buchstaben M in "PRIMT" zu bringen. Erinnern Sie sich, daß - wenn die Rechtspfeil-Taste den "cursor" über ein Zeichen auf dem Bildschirm bewegt - das Zeichen im Speicher des APPLE kopiert wird, genau so, als wenn Sie es mittels der Tastatur eingetippt hätten. Die TV-Anzeige wird nun folgendermaßen aussehen:

```
]LIST
  1Ø PRIMT "THIS IS A PREGRAM"
  2Ø GOTO 1Ø
]
```

Nun schreiben Sie den Buchstaben N, um die Schreibweise von "PRIMT" zu korrigieren. Dann gehen Sie - unter Verwendung der Rechtspfeil-Taste und der "repeat"-Taste zum Buchstaben E in "PREGRAM" über. Der TV-Schirm wird nun folgendes Aussehen haben:

```
]LIST
  1Ø PRINT "THIS IS A PREGRAM"
  2Ø GOTO 1Ø
]
```

Sollten Sie die Rechtspfeil-Taste durch zu langes Niederdrücken der "repeat"-Taste zu viele Male betätigt haben, benutzen Sie die Linkspfeil-Taste, um zum Buchstaben E zurückzugelangen. Nun schreiben Sie den Buchstaben O, um "PREGRAM" zu korrigieren und kopieren Sie unter Verwendung der Rechtspfeil-Taste bis zum Ende der Zeile 1Ø. Speichern Sie schließlich die neue Zeile in den Programmspeicher ein, indem Sie die RETURN-Taste drücken. Schreiben Sie LIST, um Ihr korrigiertes Programm sehen zu können:

```
]LIST
]
 1Ø PRINT "THIS IS A PROGRAM"
 2Ø GOTO 1Ø
]□
```

Nun lassen Sie das Programm arbeiten (RUN) (verwenden Sie "ctrl C", um das Programm zu stoppen):

```
]RUN
 THIS IS A PROGRAM
 THIS IS A PROGRAM
 THIS IS A PROGRAM
 THIS IS A PROGRAM
 THIS IS A PROGRAM
 THIS IS A PROGRAM
 THIS IS A PROGRAM
 THIS IS A PROGRAM
 BREAK IN 1Ø
]□
```

Beispiel 2 - Einsetzen von Text in eine bereits vorhandene Zeile

Nehmen Sie an, Sie wollten in das vorangegangene Beispiel ein Kommando TAB(1Ø) nach PRINT in Zeile 1Ø einfügen. Sie können das auf folgende Weise tun: Lassen Sie durch LIST die zu ändernde Zeile anzeigen:

```
]LIST 1Ø
 1Ø PRINT "THIS IS A PROGRAM"
]□
```

Tippen Sie mehrere "escape D" und ein "escape B", bis der "cursor" auf dem ersten Zeichen der zu ändernden Zeile steht. Verwenden Sie dann die Rechtspfeil- und die "repeat"-Taste, um bis zum ersten Anführungszeichen zu kopieren. (Erinnern Sie sich, ein Zeichen wird erst im Speicher kopiert, wenn Sie die Rechtspfeil-Taste betätigen, die den "cursor" von diesem Zeichen zum nächsten bewegt). Ihr TV-Display müßte nun folgendermaßen aussehen:

```
]LIST 1Ø
 1Ø PRINT"THIS IS A PROGRAM"
]
```

Nun schreiben Sie ein weiteres "escape D", um den "cursor" auf die leere Zeile genau über der zu korrigierenden zu bewegen, und das Display sieht folgendermaßen aus:

```
]LIST 1Ø
 1Ø PRINT "THIS IS A PROGRAM"
]
```

Schreiben Sie die einzusetzenden Zeichen (in diesem Fall TAB(1Ø)). Folgendes müßte nun auf dem TV-Display erscheinen:

```
]LIST 1Ø   TAB(1Ø);
 1Ø PRINT "THIS IS A PROGRAM"
]
```

Schreiben Sie ein "escape C", um den "cursor" eine Zeile tiefer zu bringen, so daß folgendes Display erscheint:

```
]LIST 1Ø   TAB(1Ø);
 1Ø PRINT "THIS IS A PROGRAM"
```

Gehen Sie nun unter Verwendung von "escape B" (die Linkspfeil-Taste würde hier die Zeichen, die Sie gerade geschrieben haben, löschen) zum ersten Anführungszeichen zurück. Das TV-Display zeigt folgendes:

```
] LIST 1Ø    TAB(1Ø);
  1Ø PRINT "THIS IS A PROGRAM"
]
```

Von hier ab kopieren Sie den Rest der Zeile unter Verwendung der Rechtspfeil- und der "repeat"-Taste, bis das Display folgendermaßen aussieht:

```
] LIST 1Ø    TAB(1Ø);
  1Ø PRINT "THIS IS A PROGRAM"
]
```

Drücken Sie die RETURN-Taste, und Sie erhalten folgendes Bild:

```
] LIST
  1Ø PRINT TAB(1Ø); "THIS IS A PR
      OGRAM"
  2Ø GOTO 1Ø
]
```

Falls Sie wünschen, das Kopieren von Extra-Leerstellen zu vermeiden, die das LIST-Format in der Mitte von Zeilen einführt (wie die zwischen dem R und dem O von PROGRAM im obigen Beispiel), verwenden Sie "escape A". Es bewegt den "cursor" nach rechts, ohne die Zeichen zu kopieren. Das kann besonders dann nützlich sein, wenn PRINT-, INPUT- und REM-Befehle kopiert werden, wo APPLESOFT Extra-Leerstellen nicht ignoriert.

Erinnern Sie sich, daß man bei Anwendung der "escape"-Tasten Text kopieren und redigieren kann, der an irgendeiner Stelle des TV-Displays dargestellt ist.

Anhang C:

Fehlermeldungen

Nachdem ein Fehler aufgetreten ist, kehrt BASIC zur Kommandoebene zurück, wie durch das Ladekennzeichen] und einen blinkenden "cursor" angezeigt wird. Variable Werte und der Programmtext bleiben intakt, aber das Programm kann nicht fortgesetzt werden, und alle GOSUB- und FOR-Schleifenzähler sind auf Ø zurückgestellt.

Um diese Unterbrechung bei einem laufenden Programm zu vermeiden, kann der allgemeine Befehl ONERR GOTO in Konjunktion mit einem Fehlerbeseitigungsprogramm verwendet werden.

Tritt ein Fehler in einem allgemeinen Befehl mit sofortiger Ausführung auf, wird keine Zeilennummer gedruckt.

Formen der Fehlermeldung:

allgemeiner Befehl mit sofortiger Ausführung	?XX ERROR
allgemeiner Befehl mit verzögerter Ausführung	?XX ERROR IN YY

In beiden angeführten Beispielen ist "XX" die Bezeichnung des spezifischen Fehlers. "YY" ist die Zeilennummer des allgemeinen Befehls mit verzögerter Ausführung, wo der Fehler aufgetreten ist. Fehler in einem allgemeinen Befehl mit verzögerter Ausführung werden erst festgestellt, wenn dieser Befehl ausgeführt wird.

Es folgen mögliche Fehlercodes und ihre Bedeutung:

CAN'T CONTINUE

Der Versuch, ein Programm fortzusetzen, wenn keines vorhanden war, oder nachdem ein Fehler aufgetreten ist, oder nachdem eine Zeile aus einem Programm gelöscht oder einem Programm hinzugefügt wurde.

DIVISION BY ZERO

Dividieren durch Null ist ein Fehler.

ILLEGAL DIRECT

Sie können die Befehle INPUT, DEF FN, GET oder DATA nicht als allgemeine Befehle mit sofortiger Ausführung verwenden.

ILLEGAL QUANTITY (ILLEGALE MENGE)

Der einer mathematischen oder Kettenfunktion zugeordnete Parameter war außerhalb des Bereiches. Fehler der ILLEGALEN MENGE können auftreten auf Grund

a) eines negativen Feldes SUBSCRIPT (z. B. LET A (-1)=Ø),
b) der Verwendung von LOG mit einem negativen oder Null-Argument,
c) der Verwendung von SOR mit einem negativen Argument,
d) von A^B mit negativem A oder nicht ganzzahligem B,
e) der Verwendung von MID$, LEFT$, RIGHT$, WAIT, PEEK, POKE, TAB, SPC, ON...GOTO oder irgendwelcher graphischer Funktionen mit einem nicht passenden Argument.

NEXT WITHOUT FOR

Die Variable in einem NEXT-Befehl entsprach nicht der Variablen in einem FOR-Befehl, der noch wirksam war, oder ein namenloses NEXT entsprach einem FOR, das noch wirksam war.

OUT OF DATA (AUßERHALB DER DATEN)

Ein READ-Befehl wurde ausgeführt, aber alle DATA-Befehle im Programm sind bereits gelesen worden. Das Programm versuchte, zu viele Daten zu lesen oder unzulängliche Daten waren im Programm enthalten.

OUT OF MEMORY (AUßERHALB DES SPEICHERS)

Folgendes kann diesen Fehler verursachen: Programm zu umfangreich, zu viele Variablen, FOR-Schleifen, verschachtelt mehr als 1Ø Stufen (levels) tief; mehr als 24 Stufen (levels) tief verschachtelte GOSUB's; ein zu komplizierter Ausdruck; mehr als 36 Stufen tief verschachtelte Parenthesen; Versuch LOMEN einzustellen: zu hoch; Versuch LOMEN einzu-

stellen: niedriger als der gegenwärtige mögliche Wert; Versuch, HIMEN einzustellen: zu niedrig.

FORMULA TOO COMPLEX (FORMEL ZU KOMPLEX)

Mehr als zwei Befehle der Form IF "XX" THEN (FALLS "XX", DANN) wurden ausgeführt.

OVERFLOW (ÜBERLAUF)

Das Ergebnis einer Berechnung war zu umfangreich, um im BASIC-Zahlenformat dargestellt zu werden. Falls das Ergebnis unterhalb des adressierbaren Bereiches liegt, wird Null als Ergebnis gegeben, und die Ausführung geht weiter, ohne daß eine Fehlermeldung ausgedruckt wird.

REDIM'D ARRAY (NEUDIMENSIONIERUNG EINES FELDES)

Nachdem ein Feld dimensioniert worden ist, wurde ein weiterer Dimensionsbefehl für das gleiche Feld festgestellt. Dieser Fehler tritt oft auf, wenn einem Feld die unrichtige Dimension 1Ø beigegeben wurde, da einem Befehl wie A(I)=3 später im Programm ein DIM A (1ØØ) folgt. Diese Fehlermeldung kann sich als nutzbringend erweisen, wenn Sie feststellen wollen, auf welcher Programmzeile ein bestimmtes Feld dimensioniert war: setzen Sie nur einen Dimensionsbefehl für dieses Feld in die erste Zeile, lassen Sie das Programm mit RUN arbeiten, und APPLESOFT sagt Ihnen, wo der ursprüngliche Dimensionsbefehl ist.

RETURN WITHOUT GOSUB (RETURN OHNE GOSUB)

Ein RETURN-Befehl wurde festgestellt, ohne daß ein entsprechender GOSUB-Befehl ausgeführt wurde.

STRING TOO LONG (KETTE ZU LANG)

Mittels des Verkettungsoperators wurde der Versuch unternommen, eine Kette von einer Länge über 255 Zeichen zu schaffen.

BAD SUBSCRIPT (SCHLECHTER INDEX)

Es wurde der Versuch gemacht, auf ein Feldelement Bezug zu nehmen, das außerhalb der Dimensionen des Feldes ist. Dieser Fehler kann auftreten, wenn die falsche Zahl der Dimensionen in einem Feldbezug verwendet wird, z. B. LET A(1,1,1)= Z, wenn A unter Verwendung von DIM A(2,2) dimensioniert worden ist.

SYNTAX ERROR (SYNTAXFFHLER)

Fehlende Parenthese in einem Ausdruck, illegale Zeichen in einer Zeil e, falsche Interpunktion usw.

TYPE MISMATCH (SCHREIBFFHLFR)

Die linke Seite einer Ergibanweisung war eine numerische Variable,und die rechte Seite war eine Kette, oder umgekehrt; oder einer Funktion, die ein Kettenargument erwartete, wurde ein numerisches Argument zugeordnet oder umgekehrt.

UNDEF'D STATFMENT (UNDEFINIERTER BFFEHL)

Es wurde versucht, eine Befehlszeilenzahl, die nicht existiert, mit GOTO, GOSUB oder THEN zu behandeln.

UNDEF'D FUNCTION (UNDFFINIFRTE FUNKTION)

Es wurde auf eine Benutzer-definierte Funktion Bezug genommen, die nicht definiert ist.

Anlage D: Platzeinsparungen

Hinweise für die Platzeinsparung

Damit Ihr Programm weniger Platz im Speicher einnimmt, könnten die folgenden Hinweise für Sie nützlich sein. Allerdings sollten die ersten beiden Methoden zur Platzeinsparung nur dann in Erwägung gezogen werden, wenn man mit einschneidenden Platzbeschränkungen konfrontiert ist. Gute Programmierer fertigen ihre Programme häufig in zwei Versionen aus, eine erweiterte und ausführlich dokumentierte (mit REMs) und eine "überarbeitete", die einen minimalen Platz im Speicher belegt.

1) Benutzen Sie Mehrfachanweisungen pro Zeile. Es gibt eine kleine Menge von informationslosen Bytes (5Bytes), die mit jeder Zeile Ihres Programms verbunden sind. Zwei dieser 5 Bytes enthalten die Zeilennummer der Zeile im Binärsystem. Das bedeutet, daß unabhängig von der Menge der Stellen in Ihrer Zeilennummer (die minimale Zeilennummer ist Ø, die maximale 65529) immer eine gleiche Anzahl von Bytes (zwei) benötigt wird. Je mehr Anweisungen Sie auf eine jede Zeile bringen, desto geringer wird die Anzahl der Bytes sein, die von Ihrem Programm in Anspruch genommen wird. (Eine einzelne Zeile kann aus bis zu 239 Zeichen bestehen.)

Beachten Sie: Das Verbinden vieler Anweisungen auf einer Zeile erschwert die Überarbeitung und andere Veränderungen. Das Programm läßt sich außerdem schwerer lesen und verstehen, nicht nur für andere, sondern auch für Sie selbst, wenn Sie auf das Programm später zurückkommen.

2) Löschen Sie alle REM-Anweisungen. Jede REM-Anweisung nimmt mindestens ein Byte zusätzlich zu der Anzahl der Byte im gewöhnlichen Text in Anspruch. So verbraucht die Anweisung 13Ø REM THIS IS A COMMENT 24 Bytes des Speichers. In der Anweisung

14Ø X = X + Y : REM UPDATE SUM

verbraucht das REM 12 Bytes des Speichers einschließlich des Doppelpunktes vor dem REM.

Beachten Sie: Wie die Programme, die sich über mehrere Zeilen erstrecken, sind auch Programme ohne detaillierte REM-Anweisungen sehr schwer zu lesen und zu verstehen, nicht nur für andere sondern auch für Sie, wenn Sie später auf das Programm zurückkommen.

3) Verwen den Sie, wo immer es möglich ist, ganzzahlige Bereiche an Stelle von Bereichen reeller Zahlen. (Siehe auch Informationen zur Speicherzuweisung, weiter unten in der Anlage.)

4) Verwenden Sie Variable an Stelle von Konstanten. Nehmen wir an, daß sie die Konstante 3.14159 zehnmal in Ihrem Programm verwenden.
Wenn Sie eine Anweisung

1Ø P = 3.14159

in das Programm einfügen und P an Stelle von 3.14159 verwenden, wenn die Konstante benötigt wird, dann werden Sie 40 Bytes sparen.
Außerdem wird sich daraus eine Erhöhung der Geschwindigkeit ergeben.

5) Ein Programm muß nicht mit einem END beendet werden. Deshalb kann eine derartige Anweisung am Ende des Programms gelöscht werden.

6) Verwenden Sie die gleichen Variablen wieder. Wenn Sie die Variable T nehmen, um in einem Teil des Programms ein zeitweiliges Resultat festzuhalten, und Sie eine zeitweilige Variable weiter unten in Ihrem Programm benötigen, dann nehmen Sie diese wieder. Oder, wenn Sie den Nutzer des Computers bitten, auf zwei verschiedene Fragen zu zwei verschiedenen Zeitpunkten im Verlaufe des Programmlaufs mit Ja oder Nein antworten, dann verwenden Sie die gleichen zeitweiligen Variablen A$, um die Antwort zu speichern.

7) Benutzen Sie GOSUB, um Teile von Programmanweisungen ausführen zu lassen, bei denen gleichartige Operationen ablaufen.

8) Benutzen Sie die Null-Elemente von Matrizen, z.B. A(Ø), B(Ø,X).

9) Wenn für A$="CAT" eine Neuzuweisung A$="DOG" erfolgt, wird die alte Gruppe "CAT" nicht aus dem Speicher gelöscht. Wenn Sie eine Anweisung in der Form

X = FRE(Ø)

periodisch in Ihr Programm einbauen, dann wird durch APPLESOFT jede alte Gruppe aus dem Speicher "herausgeworfen".

Informationen über die Speicherzuweisung

Einfache reelle, ganzzahlige oder Gruppenvariablen wie V, V% oder V$ brauchen 7 Bytes. Reelle Variablen brauchen 2 Bytes für den Variablennamen und 5 Bytes für den Wert (1 Exponent, 4 Mantisse). Die ganzzahligen Variablen brauchen 2 Bytes für den Namen, 2 Bytes für den Wert und haben Ø (Nullen) in den verbleibenden drei Bytes. Die Stringvariablen benötigen 2 Bytes für den Variablennamen, 1 Byte für die Länge der Gruppe, 2 Bytes für einen Stellenzeiger für den String im Speicher, und die restlichen 2 Bytes werden mit Nullen belegt. Eine Speichertabelle befindet sich auf Seite 205.

Die reellen Bereichsvariablen benötigen mindestens 12 Bytes, nämlich zwei für den Namen, zwei für die Größe des Bereichs, eines für die Anzahl der Dimensionen, zwei für die Größe einer jeden Dimension und fünf für jedes Bereichselement. Ganzzahlige Bereichsvariablen brauchen nur 2 Bytes für jedes Bereichselement. Stringbereichsvariablen benötigen 3 Bytes für jedes Bereichselement, nämlich eines für die Länge und zwei für einen Zeiger (Siehe Seite)

Stringvariable, einfache oder Bereichsvariable, benötigen ein Byte Speicherraum für jedes Zeichen des Strings. Den String selbst wird in der Reihenfolge des Auftretens im Programm beginnend bei HIMEM: ein Speicherplatz zugeordnet.

Wenn eine neue Funktion durch eine DEF-Anweisung definiert wird, werden 6 Bytes benutzt, um den Definitionszeiger zu speichern.

Die Tabuwörter wie FOR, GOTO oder NOT und die Bezeichnungen von inneren Funktionen wie COS, INT und STR$ nehmen nur ein Byte des Programmspeichers ein. Alle anderen Zeichen in Programmen benötigen jeweils ein Byte des Programmspeichers.

Wenn ein Programm ausgeführt wird, wird auf dem Stack folgendermaßen dynamisch Platz zugewiesen:

1) Jede aktive FOR ... NEXT-Schleife benötigt 16 Bytes.
2) Jedes aktive GOSUB (eines, für das noch kein RETURN kam) benötigt 6 Bytes.
3) Jeder Klammerausdruck, der in einem Ausdruck vorkommt, braucht 4 Bytes und jedes zeitweilige Ergebnis, das in einem Ausdruck berechnet wird, benötigt 12 Bytes.

Anlage E: Beschleunigung des Programms

Die folgenden Hinweise sollten zur Verkürzung der für den Ablauf des Programms notwendigen Zeit beitragen. Beachten Sie, daß einige Hinweise die gleichen sind, wie die zur Verminderung des von Ihrem Programm belegten Speicherraumes. Das heißt, daß in vielen Fällen dort, wo die Effektivität der Speicherauslastung erhöht wird, auch der Lauf des Programms beschleunigt wird.

1) DER WAHRSCHEINLICH WICHTIGSTE HINWEIS FÜR EINE BESCHLEUNIGUNG DES PROGRAMMS UM DAS ZEHNFACHE IST DER FOLGENDE:
Benutzen Sie an Stelle von Konstanten immer Variable. Es kostet mehr Zeit, eine Konstante in ihre Gleitkomma-Darstellung (reelle Zahl) zu verwandeln als den Wert einer einfachen oder Bereichsvariablen hervorzuholen. Dies ist besonders wichtig in FOR ... NEXT-Schleifen oder irgendeinem Code, der wiederholt ausgeführt werden muß.

2) Variable, die bei der Ausführung eines BASIC-Programms das erste Mal vorkommen, werden zu Beginn der Variablenliste zugeordnet. Das heißt, daß in einer Anweisung wie

5 A = Ø : B = A : C = A

A an die erste, B an die zweite, C an die dritte Stelle der Variablenliste gesetzt wird (wir wollen annehmen, daß Zeile 5 die erste im Programm auszuführende Anweisung enthält). Im weiteren Programmlauf wird dann, wenn BASIC mit einem Verweis auf die Variable A konfrontiert wird, nur ein Eintrag in der Variablenliste gesucht, um A zu finden. Zwei Einträge müssen zur Auffindung von B, drei Einträge für das Finden von C usw. geprüft werden.

3) Benutzen Sie die NEXT-Anweisung ohne die Indexvariable. NEXT dauert nicht so lange wie NEXT I, weil nicht erst geprüft wird, ob die in NEXT angegebene Variable noch die gleiche ist wie die aus der letzten noch aktiven FOR-Anweisung.

4) Wenn im Programmablauf eine neue Zeile, die auf eine andere verweist, vorkommt, wie z.B. "GOTO 1ØØØ", dann wird das gesamte Programm des Nutzers nach der im Ver-

weis angegebenen Zeilennummer beginnend bei der mit der niedrigsten Zahl numerierten Zeile abgesucht. (Die besagte Zeilennummer ist in unserem Falle 1ØØØ.) Deshalb sollten die Zeilen, auf die häufig verwiesen wird, so früh als möglich im Programm aufgeführt werden.

Anlage F: Dezimalzeichen für Schlüsselworte

Dezimal-zeichen	Schlüsselwort	Dezimal-zeichen	Schlüsselwort
128	END	155	TRACE
129	FOR	156	NOTRACE
130	NEXT	157	NORMAL
131	DATA	158	INVERSE
132	INPUT	159	FLASH
133	DEL	160	COLOR=
134	DIM	161	POP
135	READ	162	VTAB
136	GR	163	HIMEM:
137	TEXT	164	LOMEM:
138	PR#	165	ONERR
139	IN#	166	RESUME
140	CALL	167	RECALL
141	PLOT	168	STORE
142	HLIN	169	SPEED=
143	VLIN	170	LET
144	HGR2	171	GOTO
145	HGR	172	RUN
146	HCOLOR=	173	IF
147	HPLOT	174	RESTORE
148	DRAW	175	&
149	XDRAW	176	GOSUB
150	HTAB	177	RETURN
151	HOME	178	REM
152	ROT=	179	STOP
153	SCALE=	180	ON
154	SHLOAD	181	WAIT

Dezimal-zeichen	Schlüsselwort	Dezimal-zeichen	Schlüsselwort
182	LOAD	211	INT
183	SAVE	212	ABS
184	DEF	213	USR
185	POKE	214	FRE
186	PRINT	215	SCRN (
187	CONT	216	PDL
188	LIST	217	POS
189	CLEAR	218	SOR
190	GET	219	RND
191	NEW	220	LOG
192	TAB (	221	EXP
193	TO	222	COS
194	FN	223	SIN
195	SPC (	224	TAN
196	THEN	225	ATN
197	AT	226	PEEK
198	NOT	227	LEN
199	STEP	228	STR$
200	+	229	VAL
201	-	230	ASC
202	*	231	CHR$
203	/	232	LEFT$
204	^	233	RIGHT$
205	AND	234	MID$
206	OR		
207	>		
208	=		
209	<		
210	SGN		

Anlage G: Tabuwörter im APLLESOFT

&

ABS	AND	ASC	AT	ATN			
CALL	CHR$	CLEAR	COLOR=	CONT	COS		
DATA	DEF	DEL	DIM	DRAW			
END	EXP						
FLASH	FN	FOR	FRE				
GET	GOSUB	GOTO	GR				
HCOLOR=	HGR	HGR2	HIMEM:	HLIN	HOME	HPLOT	HTAB
IF	IN#	INPUT	INT	INVERSE			
LEFT$	LEN	LET	LIST	LOAD	LOG	LOMEM:	
MID$							
NEW	NEXT	NORMAL	NOT	NOTRACE			
ON	ONERR	OR					
PDL	PEEK	PLOT	POKE	POP	POS	PRINT	PR#
READ	RECALL	REM	RESTORE	RESUME	RETURN	RIGHT$	
	RND	ROT=	RUN				
SAVE	SCALE=	SCRN(	SGN	SHLOAD	SIN	SPC(	
	SPEED=	SQR	STEP	STOP	STORE	STR$	
TAB(	TAN	TEXT	THEN	TO	TRACE		
USR							
VAL	VLIN	VTAB					
WAIT							
XPLOT	XDRAW						

Die Tabuwörter werden in APPLESOFT mit einem "Kennzeichen" belegt. Jedes dieser Wörter belegt dann nur 1 Byte im Programmspeicher. Alle anderen Zeichen im Programmspeicher benötigen jeweils 1 Byte des Progammspeichers. Siehe Anlage F zu den Kennzeichen der Tabuwörter.

ACHTUNG!

Das &-Zeichen ist nur für computer-internen Betrieb gedacht. Es ist kein richtiger APPLESOFT-Befehl. Wenn es als Anweisung ausgeführt wird, verursacht das Symbol einen unbedingten Sprung auf den Speicherplatz $3F5. Für die Umschaltung benutzen Sie reset ctrl C return.

ACHTUNG!

XPLOT ist ein Tabuwort, das keinem laufenden APPLESOFT-Befehl entspricht.
Einige Tabuwörter werden durch APPLESOFT nur in bestimmten Kontexten als solche anerkannt.

COLOR, HCOLOR, SCALE, SPEED und ROT

werden nur als Tabuwörter identifiziert, wenn das nächste sich ohne Abstand anschließende Zeichen das Ersatzzeichen = ist. Im Falle von COLOR und HCOLOR ist das von geringem Nutzen, da das in ihnen enthaltene Tabuwort OR ihre Verwendung in Variablennamen ohnehin unterbindet.

SCRN, SPC und TAB

werden nur als Tabuwörter identifiziert, wenn das nächste, sich ohne Abstand anschließende Zeichen die nach rechts geöffnete Klammer ist.

HIMEM:

muß den Doppelpunkt besitzen, um als Tabuwort identifiziert zu werden.

LOMEM:

braucht auch den Doppelpunkt für die Identifizierung als Tabuwort.

ATN

wird nur als Tabuwort identifiziert, wenn zwischen dem T und dem N kein Zwischen-

raum vorhanden ist. Sollte ein Zwischenraum vorhanden sein, dann wird das Tabuwort AT identifiziert und nicht ATN.

TO wird als Tabuwort identifiziert, es sei denn ein A geht voraus und zwischen T und O liegt ein Zwischenraum. Ist das der Fall, dann wird das Tabuwort AT an Stelle von TO identifiziert.

Manchmal kann man Klammern benutzen, um Tabuwörtern aus dem Wege zu gehen. So wird

1ØØ FOR A = LOFT OR CAT TO 15

aufgelistet als 1ØØ FOR A = LOF TO RC AT TO 15, aber

1ØØ FOR A = (LOFT) OR (CAT) TO 15

wird aufgelistet als 1ØØ FOR A = (LOFT) OR (CAT) TO 15.

Anlage H: Umwandlung von BASIC-Programmen zu APPLESOFT

Obwohl sich die praktische Anwendung von BASIC auf verschiedenen Computern in vielem ähnelt, gibt es einige Unverträglichkeiten, auf die Sie achten sollten, wenn Sie planen, BASIC-Programme in APPLESOFT zu konvertieren.

1) Bereichs(Matrix)-Index: Einige BASICs verwenden " [" und "] " für die Kennzeichnung von Bereichsindizes. APPLESOFT verwendet " (" und ") ".

2) Strings: In einer Reihe von BASICs sind Sie gezwungen, die Länge der Strings zu dimensionieren (erklären), bevor sie benutzt werden. Sie sollten alle diese Dimensions-Anweisungen aus dem Programm streichen. In einigen dieser BASICs dimensioniert eine Ankündigung in der Form DIM A$ (I, J) einen Stringbereich von J Elementen, von denen jedes die Länge I hat. Wandeln Sie diese DIM-Anweisungen in das APPLESOFT-Äquivalent DIM A$(J) um.

Bei APPLESOFT werden Gruppenverkettungen mit "+" gekennzeichnet, und nicht durch "," oder "&".

APPLESOFT verwendet LEFT$, RIGHT$ und MID$, um aus Strings Unterstrings herauszulösen. Andere BASICs verwenden A$(I), um Zugang zum I-ten Zeichen des Strings A$ zu erlangen, und A$(I, J), um einen Unterstring aus A$, beginnend mit der I-ten Zeichenposition bis zur J-ten Zeichenposition, herauszunehmen.

Wandeln Sie folgendermaßen um:

alt	neu
A$(I)	MID$(A$, I, 1)
A$(I, J)	MID$(A$, I, J-I+1)

Dabei wird angenommen, daß der Bezug auf einen Unterstring von A$ in einem Ausdruck oder rechts von einer Zuordnung vorgenommen wird. Wird der Bezug auf A$ links von der Zuordnung vorgenommen und ist X$ der Stringausdruck, der die Zeichen in A$ ersetzen soll, dann wandeln Sie folgendermaßen um

alt	neu
A$(I) = X$	A$= LEFT$(A$, I-1)+ X$+ MID$(A$, I+1)
A$(I, J)= X$	A$= LEFT$(A$, I-1)+X$+ MID$(A$, J+1)

3) In einigen BASICs sind "Mehrfach-Zuordnungen" zulässig, die folgende Form haben:

```
5ØØ LET B = C = Ø
```

Diese Anweisung würde sowohl Variable B als auch C auf Null setzen.

In APPLESOFT-BASICwirkt sich das vollkommen anders aus. Alle Gleichheitszeichen rechts vom ersten würden als logische Vergleichsoperatoren ausgelegt. Das würde die Variable B auf -1 setzen, wenn C = Ø. Wenn C ≠ Ø, würde B auf Ø gesetzt.

Am einfachsten lassen sich solche Anweisungen umwandeln, wenn man sie folgendermaßen umschreibt:

```
5ØØ C = Ø : B = C
```

4) Bei einigen BASICs wird "/" an Stelle von ":" verwendet, um Mehrfachanweisungen pro Zeile zu trennen. Verwandeln Sie also alle "/" in ":" im Programm.

5) Programme, bei denen mathematische Funktionen verwendet werden, die in einigen BASICs verfügbar sind, müssen unter Verwendung von FOR ... NEXT-Schleifen neu geschrieben werden, damit die entsprechenden Operationen ausgeführt werden können.

Anlage I: Speicherraumtabelle

Speicherbereich	Beschreibung
Ø.1FF	Arbeitsraum für Programm, nicht verfügbar für Benutzer
2ØØ.2FF	Pufferspeicher für Tastenfeldzeichen
3ØØ.3FF	Bereich verfügbar für kurze maschinensprachliche Programme des Nutzers
4ØØ.7FF	Bildschirmanzeige für Seite 1 der Text- und Farbgrafikdarstellung
8ØØ.2FFF	APPLESOFT-BASIC-Übersetzer bei der Kassettentonbandversion
8ØØ.XXX	Bei der Installierung von APPLESOFT-Firmware (Teilnummer: A2BØØØ9X) Raum für Programme und Variablen des Benutzers, wobei XXX der maximale RAM-Speicher ist, der durch APPLESOFT verwendet werden kann. Das ist entweder der totale RAM-Speicher des Systems oder etwas weniger, falls der Benutzer einen Teil des oberen Speichers für maschinensprachliche Programme oder als Bildschirm-Pufferspeicher bei dem Betrieb mit hoher Auflösung nutzen will.
2ØØØ.3FFF	Nur APPLESOFT-Firmware: Anzeige für Hochauflösungsgrafik auf Seite 1
3ØØØ.XXX	Kassettentonband APPLESOFT II: Programme und Variablen des Benutzers, wobei XXX der maximale RAM-Speicher ist, der von APPLESOFT benutzt werden kann. Das ist entweder der gesamte RAM-Speicher des Systems oder etwas weniger, falls der Benutzer einen Teil des oberen Speichers für maschinensprachliche Programme oder für Hochauflösungsgrafik der Seite 2 reserviert.
4ØØØ.5FFF	Anzeige von Hochauflösungsgrafik der Seite 2
CØØØ.CFFF	E/A-Adressen der Hardware

Speicherbereich	Beschreibung
DØØØ.DFFF	zukünftige ROM-Erweiterung
DØØØ.F7FF	APPLESOFT-II-Firmware-Version, Auswahlschalter in "ON"-Stellung (nach oben)
EØØØ.F7FF	Ganzzahliger BASIC auf APPLE
F8ØØ.FFFF	Monitor des APPLE-Systems

Diagramm der Programmspeicherraumtabelle von APPLESOFT

Kassetten-version	Zeiger	Firmware-version
	Plattenbetriebssystem (wenn Platten verwendet werden)	
	$73 - $74 (HIMEM:) HIMEM: wird automatisch auf den höchsten RAM-Speicherplatz gesetzt, sofern es nicht vom Nutzer vorgegeben wurde.	
	Strings $6F - $7Ø	
	Freier Speicherraum einschließlich Bildschirmpufferspeicher für Hochauflösungsgrafik (bei Kassetten-APPLESOFT nur Seite 2 verfügbar) Beachte: String-Bereich kann sich mit alten Angaben füllen und in den Hochauflösungsgrafikspeicher oder in den Maschinenprogrammspeicher überlaufen. Zur Vermeidung	

Kassettenversion	Zeiger	Firmwareversion
	dieses Vorkommnisses und zur Einleitung der Speicherreinigung fügt man in das Programm X=FRE(Ø) ein.	
	$6D - $6E numerische und Stringzeigerbereiche (Siehe Anlage Y unter Variablenkarte) $6B - $6C	
	einfache Variablen (siehe Anlage J unter Variablenkarte) $69 - $6A (LOMEM:)	
$3ØØ1	$AF - $BØ Programm $67 - $68	$8Ø1
$2FFF $8Ø1	APPLESOFT	F7FF DØØØ

Anlage J: PEEK-, POKE- und CALL-Befehle

Hier nun einige der speziellen Eigenschaften von APPLESOFT, die Sie mit Hilfe von PEEK-, POKE- oder CALL-Befehlen nutzen können. Achten Sie darauf, daß einige davon die Auswirkungen anderer Befehle von APPLESOFT duplizieren.

Einfache Schaltprozesse sind normalerweise abhängig von der Adresse: Jeder Befehl, der diese Adresse beinhaltet, hat die gleiche Wirkung auf den Schalter. Somit kann man folgendes Beispiel nennen:

POKE - 163Ø4,Ø

Aber dieselbe Wirkung kann man erzielen, wenn man POKE für diese Adresse mit jeder beliebigen Zahl von 0 bis 255 oder PEEK für diese Adresse eingibt:

X= PEEK (- 163Ø4)

Das trifft nicht für Befehle zu, bei denen Sie für die angeforderte Adresse POKE mit einem spezifischen Wert eingeben müssen, womit ein Rand gesetzt und der Cursor auf einen speziellen Platz transportiert wird.

EINSTELLUNG DES TEXTSCHREIBFELDES

Die ersten vier POKE-Kommandos mit den Zeilennummern 10, 20, 30 und 40 als Beispiel dienen dazu, die Größe des "Fensters" einzustellen, in dem der Text auf dem Bildschirm gezeigt und gedruckt wird. Damit wird jeweils der linke Rand, die Zeilenbreite, der obere und der untere Rand des Textfeldes gesetzt.

Durch das Setzen des Textfeldes wird das Verbleibende auf dem Bildschirm nicht gelöscht und der Cursor nicht in das Textfeld transportiert (benutzen Sie HOME oder HTAB und VTAB). Der VTAB-Befehl ignoriert das Textfeld vollkommen: der oberhalb des Textfeldes gedruckte Text erscheint normal, während der gesamte Text, der unterhalb dieses Fensters gedruckt wird, auf einer Zeile erscheint. HTAB kann außerdem den Cursor außerhalb des Textfeldes transportieren, aber nur soweit, um ein Zeichen dort zu drucken.
Eine Veränderung der Zeilenbreite wirkt sich sofort aus, wogegen eine Änderung des linken Randes erst dann bemerkt wird, wenn der Cursor versucht, zu dem linken Rand "zurückzukehren".

ACHTUNG!

Der auf dem Bildschirm angezeigte Text ist lediglich ein spezielles Abbild eines besonderen Teiles des Speichers von APPLE (Text, Seite 1). Für die Textanzeige "blickt" der Bildschirm immer auf denselben Teil des Speichers, um zu sehen, was der Computer dort "geschrieben" hat. Wenn das Textfeld verändert wird, wird dem Computer mitgeteilt, wo er den Text in dem Speicher "schreiben" soll. Das funktioniert solange, wie ein Speicherteil innerhalb des gewöhnlichen Textbereiches spezifiziert ist. Wenn Sie den linken Rand angenommen auf 255 einstellen (das Maximum sollte 4Ø sein, da der Bildschirm eine Breite von 4Ø Druckpositionen hat), befehlen Sie dem Computer, den Text weit über den normalen, für den Text reservierten Speicherbereich hinaus zu "schreiben". Dieser Speicher wird nicht auf dem Bildschirm gezeigt, kann aber Teile Ihres Programms oder sogar die für das APPLESOFT notwendigen Informationen enthalten. Um Ihr Programm und das APPLESOFT zu sichern, setzen Sie das Textfeld nicht außerhalb der Grenzen des 24 Zeilen a'4Ø Zeichen umfassenden Bildschirms.

1Ø POKE 32, L

setzt den linken Rand der Bildschirmanzeige auf den durch L festgelegten Wert, der in dem Bereich von Ø bis 39 liegt, wobei Ø die äußerst linke Position darstellt. Diese Veränderung wird erst dann realisiert, wenn der Positionsanzeiger versucht, zu dem linken Rand "zurückzukehren".

ACHTUNG!

Die Breite des Textfeldes wird nicht durch diesen Befehl geändert, was bedeutet, daß der rechte Rand in dem selben Maße wie der linke verschoben wird. Um Ihr Programm und das APPLESOFT zu erhalten, verringern Sie zuerst die Breite des Fensters in entsprechendem Maße und verändern dann den linken Rand.

2Ø POKE 33, W

Mit diesem Befehl wird die Breite (Anzahl der Zeichen pro Zeile) der Bildschirmanzeige auf den durch W festgelegten Wert eingestellt, der in dem Bereich von 1 bis 4Ø liegt.

VERBOTEN!

Stellen Sie W nicht auf Null: POKE 33,∅ ruiniert APPLESOFT.

ACHTUNG!

Wenn W kleiner ist als 33, können in der dritten Tabellenspalte des PRINT-Befehls auch Zeichen außerhalb des Textfeldes gedruckt werden.

3∅ POKE 34, T

Damit wird der obere Rand der Bildschirmanzeige auf den durch T festgelegten Wert gesetzt, der in dem Bereich von∅ bis 23 liegt, wobei ∅ die oberste Zeile des Bildschirms ist. Der Befehl POKE 34,4 gestattet nicht, daß der Text auf den ersten vier Zeilen des Bildschirms ausgedruckt wird. Setzen Sie den oberen Rand des Textfeldes (T) nicht unter den unteren Rand (B, unten).

4∅ POKE 35, B

Damit wird der untere Rand der Bildschirmanzeige auf den durch B festgelegten Wert eingestellt, der in dem Bereich von ∅ bis 24 liegt, wobei 24 die untere Zeile auf dem Bildschirm ist. Setzen Sie den unteren Rand des Textfeldes (B) nicht höher als den oberen Rand (T, oben).

ANDERE BEFEHLE, DIE DEN TEXT, DAS TEXTFELD UND DAS TASTENFELD BEEINFLUSSEN

45 CALL -936

Damit werden alle Zeichen innerhalb des Textfeldes gelöscht und der Positionsanzeiger auf die äußerste linke Druckposition des Textfeldes transportiert. Das ist genauso wie bei esc return (Escape @) und dem HOME-Befehl.

5∅ CALL -958

Der Befehl löscht alle Zeichen innerhalb des Textfeldes angefangen bei der gegenwärtigen Cursorposition bis hin zu dem unteren Rand. Zeichen oberhalb des Positionsanzeigers sowie Zeichen, die sich in der Druckzeile links von dem Positionsanzeiger befinden, werden nicht

beeinflußt. Das ist wie bei esc F (Escape F).
Befindet sich der Cursor oberhalb des Textfeldes, wird alles vom Cursor bis zum rechten, linken und unteren Rand hin gelöscht, so als wäre der obere Rand über dem Cursor. Normalerweise ist es nicht wünschenswert, diesen Befehl zu verwenden, wenn sich der Positionsanzeiger unter dem unteren Rand des Textfeldes befindet. Normalerweise wird die untere Zeile des Textfeldes sowie die Zeile, in der sich der Positionsanzeiger befindet, über die gesamte Breite des Textfeldes gelöscht.

6Ø CALL -868
Mit diesem Befehl wird die laufende Zeile vom Cursor bis zum rechten Rand gelöscht.
Das ist genauso wie bei esc E (Escape E).

7Ø CALL -922
erteilt den Befehl zum Zeilenvorschub, genauso wie bei ctrl J (Control J).

8Ø CALL -912
Mit diesem Befehl wird der Text um eine Zeile gehoben, d.h. jede Zeile des Textes wird innerhalb des begrenzten Textfeldes um eine Position höher transportiert. Dabei geht die alte obere Zeile verloren und die alte zweite Zeile wird zur ersten und die ehemalige untere Zeile ist jetzt frei. Zeichen außerhalb des Textfeldes werden nicht verändert.

9Ø X = PEEK (-16384)
Durch den Befehl wird das Tastenfeld gelesen. Wenn X >127, dann ist eine Taste gedrückt worden, und X ist der ACII-Wert der betätigten Taste, wobei das Bit 7 gesetzt wurde (eins). Das eignet sich in langen Programmen, bei denen der Computer überprüft, ob der Benutzer eine Unterbrechung zur Eingabe neuer Daten vornehmen will, ohne die Programmausführung anzuhalten.

1ØØ POKE -16368,Ø
Damit wird der Tastenfeldtakt auf Null gesetzt, so daß neue Zeichen eingelesen werden können. Das sollte sofort nach dem Lesen der Tastatur getan werden.

BEFEHLE, DIE DEN POSITIONSANZEIGER BETREFFEN

110 CH = PEEK(36)

liest die laufende horizontale Position des Positionsanzeigers und setzt die Variable CH ihr gleich. CH liegt in dem Bereich von 0 bis 39 und ist die Position des Cursors in Bezug auf den linken Rand des Textfeldes wie er durch POKE 32,L gesetzt wurde. Wenn der linke Rand mit POKE 32,5 eingestellt wurde, ist somit das äußerste linke Zeichen in dem Textfeld auf der 6. Druckposition, von der linken Randbegrenzung des Bildschirms gerechnet und wenn auf PEEK(36) hin 5 wiedergegeben wurde, befand sich der Cursor auf der elften Druckposition, von der linken Randbegrenzung des Bildschirmes gerechnet und auf der sechsten Druckposition vom linken Rand des Textfeldes aus. (Auf den ersten Blick scheint das verwirrend zu sein, denn die äußerste linke Position ist Null und nicht 1.) Das ist mit der Funktion POS(X) identisch. (Siehe nächstes Beispiel)

120 POKE 36,CH

Damit wird der Cursor auf die Druckposition CH+1, vom linken Rand des Textfeldes aus gezählt, transportiert. (Beispiel: POKE 36,0 veranlaßt, daß das nächste Zeichen am linken Rand des Textfeldes ausgedruckt wird.) Wenn der linke Rand des Textfeldes auf 6 (POKE 6) eingestellt worden ist und Sie ein Zeichen schon drei Positionen von der Randbegrenzung des Bildschirms entfernt bereitstellen wollen, ist es notwendig, den linken Rand des Fensters vor dem PRINT-Befehl zu verändern. CH muß kleiner als oder genauso groß wie die Textfeldbreite, wie sie mit POKE 22,W festgelegt wurde, und größer als oder gleich 0 sein. Dieser Befehl kann ebenso wie HTAB den Positionsanzeiger über den rechten Rand des Textfeldes hinaus bewegen, aber nur solange, um ein Zeichen zu drucken.

130 CV = PEEK (37)

Damit wird die laufende vertikale Position des Läufers gelesen und CV ihr gleich gesetzt. CV stellt die absolute vertikale Position des Läufers dar und bezieht sich nicht auf den oberen oder unteren Rand des Textfeldes. Das bedeutet, daß CV=0 die obere und CV=23 die untere Zeile des Bildschirms ist.

140 POKE 37,CV
verschiebt den Cursor auf die absolute durch CV festgelegte vertikale Position. 0 ist die oberste Zeile und 23 die untere.

KOMMANDOS FÜR GRAFISCHE DARSTELLUNGEN

Um sowohl Text als auch Grafiken anzeigen zu können, ist der Speicher des APPLE in vier Bereich unterteilt, nämlich in Textseite 1 und 2 sowie in HR-Grafikseiten 1 und 2 (grafische Darstellungen bei hoher Auflösung).

1) Textseite 1 ist der normale Speicherbereich für alle Arten von Texten und LR-Grafiken (grafische Darstellungen bei niedriger Auflösung), der für alle TEXT- und GR-Kommandos benutzt wird.

2) Textseite 2 liegt genau über Textseite 1 im Speicher. Sie ist für den Benutzer nicht ohne weiteres zugänglich. Wie bei Textseite 1 können die auf Textseite 2 gespeicherten Informationen als Text oder als LR-Grafik interpretiert werden.

3) Die HR-Grafikseite 1 liegt im Speicher zwischen 8k und 16k. Dies entspricht dem bei HGR-Befehlen benutzten Bereich. Wenn mit dieser Seite Text geschrieben wird, dann stammt dieser von Textseite 1.

4) Die HR-Grafikseite 2 nimmt im Computer den Speicherbereich 16k bis 24k ein. Dies ist der bei HGR2-Befehlen benutzte Bereich. Wird mit dieser Seite Text geschrieben, so stammt dieser von Textseite 2.

Um die verschiedenen Grafik- und Textarbeitsweisen zu benutzen, kann man die Text- und Grafikbefehle von APPLESOFT verwenden oder diese vier verschiedenen Schalter betätigen. Wie bei vielen der hier erörterten Schalter stellt ein PEEK oder POKE auf eine bestimmte Adresse den Schalter auf eine Richtung, und ein PEEK oder POKE auf eine zweite Adresse auf die andere Richtung ein.

Mit einem Wort, diese vier Schalter wählen zwischen:

1) Textanzeige	(POKE -16303,0)
und Grafikanzeige, HR oder LR	(POKE -16304,0)
2) Seite 1 von Text oder HR-Grafiken	(POKE -16300,0)
und Seite 2 von Text oder HR-Grafiken	(POKE -16299,0)
3) Text Seite 1 oder 2 für Grafiken	(POKE -16298,0)
und HR-Grafikseite 1 oder 2 für Grafiken	(POKE -16297,0)
4) Bildschirm mit HR- oder LR-Grafikbetrieb	(POKE -16302,0)
und Bildschirm mit HR- oder LR-Grafikbetrieb und Textbetrieb	(POKE -16301,0)

150 POKE -16304,0

schaltet den Anzeigemodus von Textbetrieb auf Farbgrafikbetrieb um, ohne den Bildschirm für Grafikbetrieb unter Bildung einer schwarzen Grundfläche freizumachen. Je nach den Einstellungen der anderen drei Schalter können die Grafiken, auf die umgeschaltet wurde mit geringer Auflösung oder hoher Auflösung von Seite 1 oder 2 und entweder auf dem Bildschirm mit Grafik- und Textbetrieb oder auf dem Bildschirm mit ausschließlichen Grafikbetrieb dargestellt werden.
Ähnliche APPLESOFT-Kommandos: Das GR-Kommando schaltet auf LR- Seite 1 und die in Grafik- und Textbetrieb unterteilte Anzeige und löscht die Grafikanzeige bei gleichzeitigem Erzeugen einer schwarzen Grundfläche. Das HGR-Kommando schaltet auf HR-Seite 1 und die in Grafik- und Textbetrieb unterteilte Anzeige und löscht die Grafikanzeige bei gleichzeitiger Erzeugung einer schwarzen Grundfläche. Das HGR2-Kommando schaltet auf HR-Seite 2 und die Bildschirmanzeige mit ausschließlichem Grafikbetrieb und löscht den - gesamten Bildschirm unter Bildung einer schwarzen Grundfläche.

160 POKE -16303,0

schaltet den Anzeigemodus von beliebiger Farbgrafikanzeige auf Textmodus um, ohne das Textfeld zum Heben der Zeilen um eine Position zurückzustellen. Je nach der Stellung des Schalters Seite 1/Seite 2 kann die Textseite, auf die geschaltet wurde, entweder die Textseite 1 oder die Textseite 2 sein.

Das TEXT-Kommando schaltet vollkommen auf Textbetrieb um, wählt aber zusätzlich die Textseite 1 aus, setzt das Textfeld zum Heben der Zeilen zurück auf das Maximum und positioniert den Cursor in der linken unteren Ecke des Bildschirms.

170 POKE -16302,0

schaltet von der in Grafik- und Textbetrieb unterteilten Bildschirmanzeige auf die Anzeige mit ausschließlichem Grafikbetrieb.

Je nach der Stellung der anderen Schalter können dabei Text, LR-Grafiken auf einem Raster von 40 x 48 oder HR-Grafiken auf einem Raster von 278 x 192 erscheinen.

180 POKE -16301,0

schaltet von dem Bildschirm mit ausschließlichem Grafikbetrieb auf den Bildschirm mit Grafik- und Textbetrieb um, bei dem sich unten auf dem Bildschirm vier Zeilen mit 40 Zeichen befinden.

Je nach den Einstellungen der anderen Schalter können sich auf dem oberen Teil des Bildschirms Text, LR-Grafiken auf einem Raster von 40 x 40 oder HR-Grafiken auf einem Raster von 278 x 160 befinden. Beide Teile der Bildschirmanzeige stammen von der selben Seitenzahl (1 oder 2).

184 POKE -16300,0

schaltet von Seite 1 auf Seite 2, ohne den Bildschirm zu löschen oder den Cursor zu verschieben. Dieser Befehl wird notwendig, wenn man von APPLESOFT auf Integer-BASIC übergeht, sonst würde man noch auf Seite 2 des Speichers blicken.

Je nach den Einstellungen der anderen Schalter kann damit die Umschaltung der Anzeige von HR-Grafikseite 2 auf HR-Grafikseite 2 auf HR-Grafikseite 1, von LR-Grafikseite 2 auf LR-Grafikseite 1 oder von Textseite 2 auf Textseite 1 veranlaßt werden.

186 POKE -16299,0

Mit diesem Befehl wird von Seite 1 auf Seite 2 umgeschaltet, ohne die Anzeige zu löschen oder den Cursor zu verschieben.

Je nach den Einstellungen der anderen Schalter kann damit die Umschaltung der Anzeige von HR-Grafikseite 1 auf HR-Grafikseite 2, von LR-Grafikseite 1 auf LR-Grafikseite 2 oder von

Textseite 1 auf Textseite 2 veranlaßt werden.

190 POKE -16298,0

Dieser Befehl schaltet die Seite für grafische Darstellungen von einer HR-Grafikseite auf eine gleiche Textseite um, ohne den Bildschirminhalt zu löschen. Dies ist notwendig, wenn man von APPLESOFT in Integer-BASIC geht. Andernfalls wird auf die GR-Anweisung des Integer-BASIC fälschlicherweise die HR-Seite gezeigt.
Je nach der Einstellung der anderen Schalter kann das die Umschaltung der Anzeige von HR-Grafikseite 1 auf LR-Grafikseite 1 oder von HR-Grafikseite 2 auf LR-Grafikseite 2 bewirken. Im Textbetrieb muß es zu keiner Veränderung der Anzeige kommen.

195 POKE -16297,0

schaltet die Seite für grafische Darstellungen von einer Textseite auf die entsprechende HR-Seite um, ohne dabei den Bildschirminhalt zu löschen.
Je nach Stellung der anderen Schalter kann das die Veränderung der Anzeige von LR-Grafikseite 1 auf HR-Grafikseite 1 oder von LR-Grafikseite 2 auf HR-Grafikseite 2 bewirken. Im Text muß es zu keiner Veränderung der Anzeige kommen.

200 CALL -1994

löscht die oberen 20 Zeilen der Textseite 1 bei gleichzeitigem Ausdrucken von @ -Umkehrzeichen. Im LR-Grafikbetrieb auf Seite 1 werden damit die oberen 40 Zeilen der Grafikbildschirmanzeige bei gleichzeitiger Erzeugung eines schwarzen Untergrundes freigemacht. Der Befehl hat keinen Einfluß auf Textseite 2 oder HR-Grafikseite.

205 CALL -1998

löscht die gesamte Textseite 1 bei gleichzeitigem Ausdrucken von @-Umkehrzeichen. Bei ausschließlichem LR-Grafikbetrieb (ohne Textanzeige) auf Seite 1 wird der gesamte Bildschirminhalt gelöscht, und eine schwarze Bildfläche entsteht. Der Befehl hat keine Auswirkungen auf Textseite 2 oder HR-Grafikseite.

2ØØ CALL 6245Ø
Der gegenwärtige HR-Bildschirminhalt wird gelöscht (APPLESOFT hält den zuletzt benutzten Bildschirminhalt fest, unabhängig von der Schalterstellung), und es entsteht eine schwarze Bildfläche.

21Ø CALL 62454
Der gegenwärtige HR-Bildschirminhalt wird gelöscht (APPLESOFT hält den zuletzt benutzten Bildschirminhalt unabhängig von der Schalterstellung fest), und es entsteht eine Bildfläche in der Farbe, wie sie durch HCOLOR in dem letzten HPLOT-Befehl vorgegeben wurde. Diesem Befehl muß ein gesetztes Zeichen vorausgehen.

KOMMANDOS FÜR DIE SPIELREGLER UND DEN LAUTSPRECHER

22Ø X = PEEK(-16336)
Einmalige Flip-Flop-Schaltung des Lautsprechers: Ein "click"-Geräusch ertönt über den Lautsprecher.

225 X = PEEK(-16352)
Einmalige Flip-Flop-Schaltung für Kassettenausgabe: Ein "click"-Geräusch wird bei der Wiedergabe der Kassette hörbar.

23Ø X = PEEK(-16287)
liest den Druckschalter auf dem Spielregler # Ø. Wenn X>127 wird dieser Knopf gerade betätigt.

24Ø X = PEEK(-16286)
dito, aber Druckschalter auf dem Spielregler # 1.

25Ø X = PEEK(-16285)
Druckknopf des Spielreglers # 2.

260 POKE -16296,1

stellt das "Ankündigungs"signal des Spielreglers # 0 (Spiel E/A Kollektor, Stift 15) auf TTL-Zustand hoch (3,5V) bei offenem Kollektor. Es handelt sich dann um den "AUS"-Zustand.

270 POKE -16295,0

stellt Ausgabe des Spielreglers # 0 auf TTL-Zustand niedrig (0,3V). Das ist der "EIN-Zustand, wobei der Maximalstrom 1,6mA beträgt.

280 POKE -16294,1

stellt die Ausgabe des Spielreglers #1 (Spiel E/A Kollektor, Stift 14) auf TTL-Zustand hoch (3,5V).

290 POKE -16293,0

stellt Ausgabe des Spielreglers # 1 auf TTL-Zustand niedrig (0,3V).

300 POKE -16292,1

stellt Ausgabe des Spielreglers# 2 (Spiel E/A Kollektor, Stift 13) auf TTL-Zustand hoch (3,5V).

310 POKE -16291,0

stellt Ausgabe des Spielreglers # 2 auf TTL-Zustand niedrig (0,3V).

320 POKE -16290,1

stellt Ausgabe des Spielreglers# 3 (Spiel E/A Kollektor, Stift 12) auf TTL-Zustand hoch (3,5V).

330 POKE -16289,0

stellt Ausgabe des Spielreglers auf TTL-Zustand niedrig (0,3V).

KOMMANDOS FÜR DIE FEHLERBEARBEITUNG

340 X=PEEK (218) + PEEK (219) * 256

Diese Anweisung setzt X der Zeilennummer der fehlerhaften Anweisung gleich, falls ein ONERRGOTO-Befehl ausgeführt wurde.

350 IF PEEK (216) >127 THEN GOTO 2000

Wenn das Bit 7 auf Speicherstelle 222 (ERRFLG) wahr ist, kam es zu einer ONERRGOTO-Anweisung.

360 POKE 216,0

löscht ERRFLG, so daß normale Fehlermeldungen erscheinen.

370 Y = PEEK (222)

stellt Variable Y auf den Code ein, der den Typ des Fehlers beschreibt, der den ONERRGOTO-Sprung auslöste, Die Fehlerarten werden in der folgenden Übersicht beschrieben:

Y-Wert	Aufgetretene Fehlerart	
0	NEXT WITHOUT FOR	(NEXT ohne FOR)
16	SYNTAX	(Syntax)
22	RETURN WITHOUT GOSUB	(RETURN ohne GOSUB)
42	OUT OF DATA	(nicht in der DATA-Anweisung)
53	ILLEGAL QUANTITY	(Fehler bei der Mengenangabe)
69	OVERFLOW	(Überlauf)
77	OUT OF MEMORY	(nicht im Speicher)
90	UNDEFINED STATEMENT	(nicht definierte Anweisung)
107	BAD SUBSCRIPT	(falscher Index)
120	REDIMENSIONED ARRAY	(neu dimensionierter Bereich)
133	DIVISION BY ZERO	(Division durch Null)
163	TYPE MISMATCH	(fehlerhafte Eingabe)

Y-Wert	Aufgetretene Fehlerart	
176	STRING TOO LONG	(zu lange Kette)
191	FORMULA TOO COMPLEX	(zu komplizierte Formel)
224	UNDEFINED FUNCTION	(nicht definierte Funktion)
254	BAD RESPONSE TO AN INPUT STATEMENT	(falsche Antwort auf INPUT-Anweisung)
255	Ctrl C INTERRUPT ATTEMPTED	(Ctrl-C-Unterbrechungsversuch)

```
380 POKE 768, 1Ø4 : POKE 769, 168 : POKE 77Ø, 1Ø4 : POKE 771, 166:
    POKE 772, 223 : POKE 773, 154 : POKE 774,  72 : POKE 775, 152:
    POKE 776,  72 :  POKE 777, 96
```

Damit wird ein maschinensprachliches Unterprogramm ab Speicherplatz 768 aufgebaut, das in Fehlerbearbeitungsprogrammen benutzt werden kann. Mit dem Befehl werden einige der Schwierigkeiten bei ONERR GOTO mit Hilfe der PRINT-Anweisung und der Fehlermeldung "?OUT OF MEMORY ERROR" gelöst. Benutzen Sie den Befehl CALL 768 im Fehlerbearbeitungsprogramm.

APPLESOFT VARIABLENKARTEN

(Einfache Variablen)

ZEIGER	Reelle Zahl	Ganze Zahl	Stringzeiger
$69 - $6A	Name (pos.) 1. Byte (pos.) 2. Byte	Name (neg.) 1. Byte (neg.) 2. Byte	Name (neg.) 1. Byte (pos.) 2. Byte
	Exponent 1 Byte Mantisse m.s. Byte Mantisse Mantisse Mantisse l.s. Byte	hohes Byte niedriges Byte Ø Ø Ø	Länge 1 Byte Adresse niedriges Byte Adresse hohes Byte Ø Ø

(Stringvariablen)

ZEIGER	Reelle Zahl	Ganze Zahl	Stringzeiger
$6B - $6A	Name(pos.) 1. Byte (pos.) 2. Byte	Name (neg.) 1. Byte (neg.) 2. Byte	Name (neg.) 1. Byte (pos.) 2. Byte
	OFFSET-Zeiger für nächste Variable: Zähle zur Adresse dieses Variablennamens hinzu niedriges Byte hohes Byte	OFFSET-Zeiger für nächste Variable: Zähle zur Adresse dieses Variablennamens hinzu niedriges Byte hohes Byte	OFFSET-Zeiger für nächste Variable: Zähle zur Adresse dieses Variablennamens hinzu niedriges Byte hohes Byte
	Anzahl der Dimensionen ein Byte	Anzahl der Dimensionen ein Byte	Anzahl der Dimensionen ein Byte
	Größe der n-ten Dimension hohes Byte niedriges Byte	Größe der n-ten Dimension hohes Byte niedriges Byte	Größe der n-ten Dimension hohes Byte niedriges Byte

(Forts. Karte)

	Größe der 1. Dimension hohes Byte niedriges Byte	Größe der 1. Dimension hohes Byte niedriges Byte	Größe der 1. Dimension hohes Byte niedriges Byte
	Reelle (Ø,Ø, ... Ø) Exponent 1 Byte Mantisse m.s. Byte	Ganze% (Ø,Ø, ... Ø) hohes Byte niedriges Byte	String (Ø,Ø, ... Ø) Länge 1 Byte Adresse niedriges Byte
	Mantisse Mantisse l.s. Byte		Adresse hohes Byte
	Reelle (N,N, ... N) Exponent 1 Byte Mantisse m.s. Byte Mantisse Mantisse	Ganze% (N,N, ... N) hohes Byte niedriges Byte	String (N,N, ... N) Länge 1 Byte Adresse niedriges Byte Adresse hohes Byte
$Ø6D-$Ø6E	Mantisse l.s. Byte		

Die Strings werden in der Reihenfolge der Eingabe, von HIMEM: abwärts gespeichert. Die Stringtabelle zeigt auf das erste Zeichen eines jeden Strings und zwar am unteren Teil des im Speicher befindlichen Strings. Wenn die Strings verändert werden, werden neue Zeigeradressen geschrieben. Wenn der verfügbare Speicher voll ist, werden bei einer Speicher"reinigung" alle nicht mehr benutzten Strings gelöscht. (Die Speicher"reinigung" wird mit FRE(X) bewirkt.)

Alle Bereiche werden in langsam aufsteigender Reihenfolge mit dem äußersten rechten Index eingespeichert, d.h. daß die Zahlen in dem Bereich A%(1,1), in dem gilt A%(Ø,Ø)=Ø. A%(1,Ø)=1, A%(Ø,1)=2, A%(1,1)=3, in dem Speicher in der richtigen Reihenfolge gefunden werden können.

Anlage K: ASCII-Zeichencode

DEC = ASCII-Dezimalcode
HEX = ASCII-Hexadezimalcode
CHAR = ASCII-Name des Zeichens
n/a = kein Direktzugriff von APPLE-II-Tastatur aus

DEC	HEX	CHAR	EINGABE
Ø	ØØ	NULL	ctrl
1	Ø1	SOH	ctrl A
2	Ø2	STX	ctrl B
3	Ø3	ETX	ctrl C
4	Ø4	ET	ctrl D
5	Ø5	ENQ	ctrl E
6	Ø6	ACK	ctrl F
7	Ø7	BEL	ctrl G
8	Ø8	BS	ctrl H or ←
9	Ø9	HT	ctrl I
1Ø	ØA	LF	ctrl J
11	ØB	VT	ctrl K
12	ØC	FF	ctrl L
13	ØD	CR	ctrl M or RETURN
14	ØE	SO	ctrl N
15	ØF	SI	ctrl O
16	1Ø	DLE	ctrl P
17	11	DC1	ctrl Q
18	12	DC2	ctrl R
19	13	DC3	ctrl S
2Ø	14	DC4	ctrl T
21	15	NAK	ctrl U or →
22	16	SYN	ctrl V
23	17	ETB	ctrl W

DEC	HEX	CHAR	EINGABE
24	18	CAN	ctrl X
25	19	EM	ctrl Y
26	1A	SUB	ctrl Z
27	1B	ESCAPE	ESC
28	1C	FS	n/a
29	1D	GS	ctrl shift-M
3Ø	1E	RS	ctrl ^
31	1F	US	n/a
32	2Ø	SPACE	space
33	21	!	!
34	22	"	"
35	23	#	#
36	24	$	$
37	25	%	%
38	26	&	&
39	27	'	'
4Ø	28	(	(
41	29	)	)
42	2A	*	*
43	2B	+	+
44	2C	,	,
45	2D	-	-
46	2E	.	.
47	2F	/	/
48	3Ø	Ø	Ø
49	31	1	1
5Ø	32	2	2
51	33	3	3
52	34	4	4
53	35	5	5

DEC	HEX	CHAR	EINGABE
54	36	6	6
55	37	7	7
56	38	8	8
57	39	9	9
58	3A	:	:
59	3B	;	;
6Ø	3C	<	<
61	3D	=	=
62	3E	>	>
63	3F	?	?
64	4Ø	@	@
65	41	A	A
66	42	B	B
67	43	C	C
68	44	D	D
69	45	E	E
7Ø	46	F	F
71	47	G	G
72	48	H	H
73	49	I	I
74	4A	J	J
75	4B	K	K
76	4C	L	L
77	4D	M	M
78	4E	N	N
79	4F	O	O
8Ø	5Ø	P	P
81	51	Q	Q
82	52	R	R
83	53	S	S
84	54	T	T

DEC	HEX	CHAR	EINGABE
85	55	U	U
86	56	V	V
87	57	W	W
88	58	X	X
89	59	Y	Y
9Ø	5A	Z	Z
91	5B	[	n/a
92	5C	\	n/a
93	5D	]	(shift-M)
94	5E	^	^
95	5F	_	n/a

ASCII-Code in dem Bereich von 96 bis 255 führen zur Generierung von Zeichenauf dem Computer, die sich wie in der oben aufgeführten Liste wiederholen (zuerst die in Spalte 2 und dann die gesamte Reihe noch einmal). Obwohl CHR$(65) genau wie CHR$(193) ein A zurückgibt, identifiziert APPLESOFT die beiden nicht als das gleiche Zeichen, wenn es gruppenlogische Operatoren verwendet. Ein an den Computer angeschlossener Drucker wird die beiden Zeichen in verschiedener Weise ausdrucken.

Anlage L: Verwendung der Nullseite von APPLESOFT

Speicherplatz (in Hexaangabe)	Verwendungszweck
$Ø - $5	Sprungbefehle zur Fortsetzung des Programms in APPLESOFT (reset ØG return für APPLESOFT entspricht dem reset ctrl C return für Integer-BASIC = BASIC mit ganzen Zahlen)
$A - $C	Speicherplatz der Sprunganweisung der USR-Funktion (siehe Beschreibung USR-Funktion)
$D - $17	Allgemeine Zähler/Kennzeichen für APPLESOFT
$2Ø - $F4	Reservierte Speicherplätze für APPLE-II-Monitor
$5Ø - $61	Allgemeine Zeiger für APPLESOFT
$62 - $66	Ergebnis der letzten Multiplikation bzw. Division
$67 - $68	Zeiger des Programmanfangs, bei ROM-Version normalerweise auf $Ø8Ø1 oder bei RAM-Version (Kassettentonband) normalerweise auf $3ØØ1
$69 - $6A	Zeiger für den Anfang des Bereichs für einfache Variable, zeigt gleichzeitig die Endposition plus 1 oder 2 der Programme an, wenn nicht mit LOMEM: verändert
$6B - $6C	Zeiger für den Anfang des Raums für Bereiche
$6D - $6E	Zeiger für das Ende des benutzten numerischen Speichers
$6C - $7Ø	Zeiger für den Anfang des Speicherraums für Strings. Strings werden von diesem Platz bis zum Speicherende gespeichert.

Speicherplatz (in Hexaangabe)	Verwendungszweck
$71 - $72	Allgemeiner Zeiger
$73 - $74	Oberster-plus-1 verfügbarer Speicher für APPLESOFT. Bei Ersteingabe in APPLESOFT wird es auf den höchsten RAM-Speicherplatz gesetzt.
$75 - $76	Laufende Nummer der ausgeführten Zeile
$77 - $78	"alte Zeilennummer", durch ctrl C-, STOP- oder END-Anweisung eingestellt, gibt die Nummer der Zeile an, auf der der Programmlauf unterbrochen wurde.
$79 - $7A	"alter Textzeiger", zeigt auf den Speicherplatz der nächsten auszuführenden Anweisung.
$7B - $7C	Laufende Zeilennummer, von der DATA gelesen werden sollen.
$7D - $7E	zeigt den absoluten Speicherplatz an, von wo DATA gelesen werden.
$7F - $8Ø	Zeiger der laufenden INPUT-Quelle, bei INPUT-Anweisung auf $2Ø1 gesetzt; bei READ-Anweisung umschalten auf DATA-Anweisung im Programm, aus der gelesen wird.
$81 - $82	Hält den Wert der letzten vom Nutzer verwendeten Variablen fest.
$83 - $84	Zeiger für den Wert der letzten vom Nutzer verwendeten Variablen
$85 - S9C	Allgemeine Verwendung
$9D - $A3	Gleitkomma-Hauptakkumulator
$A4	Freie Verwendung bei mathematischen Programmen mit Gleitkomma.
$A5 - $AB	Gleitkomma-Nebenakkumulator

Speicherplatz (in Hexaangabe)	Verwendungszweck
$AC - $AE	Freie Verwendung für Kennzeichen/Zeiger
$AF - BØ	Zeiger des Programmendes (nicht verändert durch LOMEM:)
$B1 - $C8	CHRGET-Programm; APPLESOFT ruft hier jedesmal ein anderes Zeichen auf.
$B8 - $B9	Zeiger des letzten durch das CHRGET-Programm abgerufenen Zeichens
$C9 - $CD	Wahlfreie Zahl
$DØ - $D5	Hilfszeiger für HR-Grafikdarstellungen
$D8 - $DF	ONERR-Zeiger/Hilfszeiger
$EØ - $E2	X- und Y-Koordinaten in den HR-Grafiken
$E4	Farbbitgruppe für HR-Grafiken
$E5 - $E7	Freie Verwendung bie HR-Grafiken
$E8 - $E9	Zeiger des Anfangs der Formtabelle
$EA	Kollisionszähler für HR-Grafiken
$FØ - $F3	Freie Verwendung Kennzeichen
$F4 - $F8	ONERR-Zeiger

Anlage M: Unterschiede zwischen APPLESOFT und Integer BASIC

UNTERSCHIEDE BEI DEN KOMMANDOS

Die folgenden Kommandos stehen in APPLESOFT zur Verfügung, doch nicht im Integer-BASIC.

ATN						
CHR$	COS					
DATA	DEF FN	DRAW				
EXP						
FLASH	FN	FRE				
GET						
HCOLOR	HGR	HGR2	HIMEM:	HOME	HPLOT	
INT	INVERSE					
LEFT$	LOG	LOMEM:				
MID$						
NORMAL						
ON...GOSUB		ON...GOTO		ONERR GOTO		
POS						
READ	RECALL	RESTORE	RESUME	RIGHT$	ROT	
SCALE	SHLOAD	SIN	SPC	SPEED	SQR	STOP
	STORE	STR$				
TAN						
USR						
VAL						
WAIT						
XDRAW						

Die nachstehend aufgeführten Kommandos stehen für Integer-BASIC, aber nicht für APPLESOFT zur Verfügung.

AUTO
DSP
MAN MOD

Folgende Befehle werden in beiden Programmiersprachen unterschiedlich bezeichnet:

Integer BASIC	APPLESOFT
CLR	CLEAR
CON	CONT
TAB	HTAB (Beachte: in APPLESOFT gibt es auch TAB)
GOTO X 1Ø+1ØØ	ON X GOTO 1ØØ, 11Ø, 12Ø
GOSUB X 1ØØ+1ØØØ	ON X GOSUB 1ØØØ, 11ØØ, 12ØØ
CALL -936	HOME (or CALL -936)
POKE 5Ø,127	INVERSE
POKE 5Ø,255	NORMAL
X	X% (% zeigt die ganzzahlige Variable an)
#	<> or ><

WEITERE UNTERSCHIEDE

Beim Integer-BASIC wird die korrekte Syntax der Anweisung überprüft, wenn der Computer die Anweisung speichert (d.h. wenn die RETURN-Taste betätigt wird). Bei APPLESOFT erfolgt die Überprüfung bei der Ausführung der Anweisung.

Im APPLESOFT muß dem GOTO und dem GOSUB eine Zeilennummer folgen, während im Integer-BASIC eine arithmetische Variable oder ein arithmetischer Ausdruck gestattet ist.

Reelle Variablen und Konstanten ("Gleitkomma"-Zahlen mit Komma oder/und Exponent) sind in APPLESOFT zulässig, nicht aber im Integer-BASIC.
In APPLESOFT tragen nur die ersten beiden Zeichen einer Variablen Bedeutung (d.h. es werden GOOD und GOUGE als gleiche Variablen angesehen). Im Integer BASIC trägt jedes Zeichen Bedeutung.
Stringoperationen werden in den beiden Sprachen unterschiedlich definiert. Im Integer BASIC müssen sowohl Strings als auch Bereiche DIMensioniert werden, während in APPLESOFT nur die Bereiche DIMensioniert werden.
In APPLESOFT ist es zulässig, daß die Bereiche multidimensional sind. Im Integer BASIC sind sie auf eine Dimension beschränkt.

Bei der Ausführung von RUN, CLEAR oder reset ctrl B return werden alle Elemente der Bereiche in APPLESOFT nullgesetzt. Im Integer BASIC muß das im Programm des Benutzers vorgegeben werden.

Wenn im Integer BASIC die Behauptung aus einer IF...THEN-Anweisung falsch ist (d.h. = Ø), dann wird nur der THEN-Teil der Anweisung ignoriert. In APPLESOFT dagegen bleiben alle dem THEN auf der gleichen Zeile folgenden Anweisungen unberücksichtigt, wenn die IF-Behauptung falsch ist bzw. gleich Null. Das Programm springt zur nächsten Zeilennummer und setzt sich dort fort.

In APPLESOFT wird durch TRACE die Zeilennummer einer jeden einzelnen Anweisung aus einer Programmzeile, die mehrere Anweisungen enthält, angezeigt, und nicht nur die Zeilennummer der ersten Anweisung, wie das im Integer BASIC der Fall ist.

Für die CALL- PEEK- und POKE-Befehle in APPLESOFT steht der gesamte Bereich von Adressen der Speicherplätze (Ø bis 65535) zur Verfügung. Im Integer BASIC muß man sich auf Speicherplätze mit Adressen über 32767 beziehen, indem man das Zweierkomplement der negativen Werte der entsprechenden positiven Zahlen benutzt (also Platz 32768 mit -32767-1, Platz 32769 mit -32767 und 3277Ø mit -32766 usw. bezeichnet wird.)

END in einem Programm, das mit der obersten Zeilennummer endet, ist in APPLESOFT wahlfrei. Im Integer BASIC wird es aber benötigt, um in allen Fällen eine Fehlermeldung zu vermeiden.

NEXT muß im Integer BASIC mit einem Variablennamen stehen, während das in APPLESOFT wahlfrei ist.

Die Syntax der INPUT-Anweisung des Integer BASIC sieht so aus:

INPUT [String,] {var,}

Wenn var eine avar ist, dann wird durch INPUT ein ? mit oder ohne wahlfreien String ausgedruckt. Handelt es sich bei var um eine svar, wird kein ? gedruckt, ganz unabhängig davon, ob ein wahlfreier String vorhanden ist oder nicht.
In APPLESOFT hat die INPUT-Anweisung folgendes Aussehen:

INPUT [String;] {var,}

Wenn der wahlfreie String weggelassen wird, druckt APPLESOFT ein ? aus. Gibt es einen wahlfreien String, wird kein ? ausgedruckt.

ANLAGE N: Zusammenfassende Übersicht über APPLESOFT-Befehle

ABS(-3.451)

gibt den absoluten Wert eines Arguments wieder, im Beispiel 3.451 ASC("QUEST") gibt den ASCII-Dezimalcode für das erste Zeichen im Argument wieder. Im Beispiel wird 81 (ASCII für Q) wiedergegeben.

ATN(2)

gibt den Arcustangens des Arguments in Radianten wieder. Im Beispiel wird 1.1Ø714872 (Radianten) wiedergegeben.

CALL -922

löst die Ausführung eines maschinensprachlichen Unterprogramms auf der Speicherstelle aus, deren Dezimaladresse angegeben ist. Im Beispiel wird ein Zeilenvorschub realisiert.

CHR$(65)

gibt das ASCII-Zeichen wieder, das dem Wert des Arguments, welches im Bereich von Ø bis 255 liegen muß. Im Beispiel wird der Buchstabe A wiedergegeben.

CLEAR

setzt alle Variablen auf Ø und alle Gruppen auf Null.

COLOR=12

stellt die Farbe für das Abbilden von Graphen im LR-Betrieb ein. Im Beispiel heißt die Farbe grün. GR setzt die Farbe auf Ø. Die Farben und ihre dazugehörigen Zahlen lauten:

Ø - schwarz	4 - dunkelgrün	8 - braun	12 - grün
1 - magentarot	5 - grau	9 - orange	13 - gelb
2 - dunkelblau	6 - mittelblau	1Ø - grau	14 - blaugrün
3 - purpur	7 - hellblau	11 - rosa	15 - weiß

Wenn man herausfinden will, welche Farbe ein gegebener Punkt auf dem Bildschirm hat, benutzt man das SCRN-Kommando.

CONT

Wenn der Programmablauf durch ein STOP, END, ctrl C oder ein reset ØG return gestoppt wurde, wird er durch ein CONT-Kommando bei der nächsten Anweisung (wie GOSUB), nicht bei der nächsten Zeilennummer wieder aufgenommen. Es wird nichts gelöscht. Nach reset ØG return kann das Programm manchmal nicht ordentlich fortgesetzt werden, weil einige Programmzeiger und Stacks gelöscht wurden. Die CONT-Anweisung kann nicht benutzt werden, wenn

a) eine Programmzeile modifiziert, hinzugefügt oder gelöscht oder
b) eine Fehlmeldung angezeigt wurde, durch die der Programmablauf gestoppt wurde.

COS(2)

gibt den Cosinus des Arguments wieder, der im Radianten angegegen wird. Im Beispiel ergibt das - 0,415146836.

ctrl C

kann benutzt werden, um ein laufendes Programm oder eine Auflistung zu unterbrechen. Es kann auch zur Unterbrechung eines INPUT benutzt werden, wenn es als erstes Zeichen eingegeben wird. Das INPUT wird nicht eher unterbrochen, bis die RETURN-Taste betätigt wird.

ctrl X

weist APPLE II an, die gerade eingegebene Zeile zu ignorieren, ohne irgendeine vorausgehende Zeile mit der gleichen Zeilennummer zu löschen. Ein (\) wird am Ende der Zeile angezeigt, die ignoriert werden soll.

DATA JOHN SMITH, "CODE 32", 23.45, -6

bildet eine Liste von Elementen, die durch eine READ-Anweisung benutzt werden kann. In unserem Beispiel ist das erste Element das Literal JOHN SMITH, das zweite Element ist die Gruppe "CODE 32", das dritte Element ist die reelle Zahl 23.45 und das vierte Element die ganze Zahl -6.

DEF FN A(W)=2*W+W

gestattet dem Benutzer, einzeilige Funktionen in einem Programm zu definieren. Die Funktion muß zunächst definiert werden, wobei die DEF-Anweisung verwendet wird. Später im Pro-

gramm kann die vorher DEFinierte Funktion verwendet werden. Das Beispiel illustriert, wie eine Funktion FN A(W) definiert wird, die dann später im Programm in der Form FN A(23) oder FN A(-7* Q+1) usw. benutzt werden kann. FN A(23) hat zur Folge, daß 23 anstelle von W in dem Ausdruck 2 W+W geschrieben wird. Die Funktion wird dann 2 23+23 = 69 ergeben. Angenommen, Q=2, dann ist FN A(-7* Q+1) = FN A(-7* 2+1) = FN A (-13). Die Berechnung der Funktion wird dann 2*(-13)+(-13) oder -26-13 oder -39 ergeben.

DEL 23,56

Entfernt den benannten Bereich von Zeilen aus dem Programm. In unserem Beispiel werden die Zeilen 23 bis 56 gelöscht. Um eine einzelne Zeile aus dem Programm zu löschen, sagen wir Zeile 35Ø, muß der Befehl in der Form DEL 35Ø,35Ø benutzt werden. Noch einfacher ist es, die Zeilennummer einzugeben und dann die RETURN-Taste zu betätigen.

DIM AGE(2Ø,3), NAME$(5Ø)

Wenn eine DIM-Anweisung ausgeführt wird, wird Raum für die benannten Bereiche mit Indizes, die von Ø bis zum angegebenen Index reichen, reserviert, in dem Beispiel werden NAME$(5Ø) 5Ø+1 oder 51 Gruppen beliebiger Länge reserviert. Dem Bereich AGE(2Ø,3) werden (2Ø+1) *(3+1) oder 21 * 4 oder 84 reellzahlige Elemente zugewiesen. Wenn ein Bereichselement in einem Programm benutzt wrid, bevor es dimensioniert ist, wird ein maximaler Index von 1Ø für jede Dimension im Index des Elements zugeordnet. Bereichselemente werden auf Null gesetzt, wenn RUN oder CLEAR ausgeführt wird.

DRAW 4 AT 5Ø,1ØØ

zeichnet die Formativdefinition No. 4 aus einer vorher geladenen Formativtabelle im HR-Graphikbetrieb, wobei im Punlt x=5Ø, y=1ØØ begonnen wird. Die Farbe, die Drehung und der Maßstab des zu zeichnenden Formativs muß benannt werden, bevor das DRAW-Kommando ausgeführt wird.

END

veranlaßt ein Programm, die Programmausführung zu stoppen und gibt die Kontrolle an den Benutzer zurück. Es wird keine Meldung gedruckt.

esc A oder esc B oder esc C oder esc D

Die Escape-Taste kann in Verbindung mit den Buchstabentasten A, B, C oder D verwendet

werden, um den Cursor zu verschieben, ohne die Zeichen zu beeinflussen, über die der Cursor bewegt wird. Um den Cursor um eine Position zu verschieben, muß zuerst die Escape-Taste gedrückt werden, dann wird die Escape-Taste wieder freigegeben und die entsprechende Taste mit dem Buchstaben gedrückt.

Kommando	bewegt den Cursor um den Weg
esc A	rechts
esc B	links
esc C	herunter
esc D	herauf

EXP(2)
gibt den Wert von e wieder, welches in die vom Argument angegebene Potenz gehoben wurde. Bei 6 Stellen beträgt e = 2,718289, so daß in dem Beispiel 7,389∅561 wiedergegeben wird.

FLASH
stellt den Bildbetrieb auf "blinkend" ein, so daß die Ausgabe durch den Computer abwechselnd auf dem Bildschirm in weißen Zeichen auf schwarzer Grundfläche und dann umgekehrt in schwarzen Zeichen auf weißem Hintergrund erfolgt. Benutzen Sie den NORMAL-Befehl, um wieder zu einer nichtblinkenden Anzeige mit weißen Buchstaben auf schwarzem Untergrund zurückzukommen.

FOR W=1 TO 2∅ ... NEXT W
FOR Q=2 TO -3 STEP -2 ...NEXT Q
FOR 2=5 TO 4 STEP 3 ...NEXT

gestattet Ihnen eine "Schleife" zu schreiben, um beliebige Befehle zwischen dem FOR-Kommando (Spitze der Schleife) und dem NEXT-Kommando (unterster Punkt der Schleife) eine festgelegte Anzahl von Malen auszuführen. Im ersten Beispiel wird durch die Variable W angegeben, wie oft die Befehle auszuführen sind. Für die Befehle, die sich im Innern der Schleife befinden, beträgt W = 1, 2, 3, ... 2∅. Danach endet die Schleife (mit W = 21) und der Befehl nach NEXT W wird ausgeführt. Das zweite Beispiel illustriert, wie angezeigt

wird, daß die STEP-Größe, in der gezählt wird, ungleich 1 ist. Die Prüfung findet am Ende der Schleife statt. So werden im dritten Beispiel die Befehle innerhalb der Schleife einmal ausgeführt.

FRE(Ø)
gibt die dem Benutzer noch verfügbare Speichermenge in Bytes wieder. Was Sie in die Klammern schreiben, ist solange unbedeutend, wie APPLESOFT es auswerten kann.

GET ANS $
holt ein einzelnes Zeichen aus dem Tastenfeld, ohne es auf dem Bildschirm zu zeigen und ohne die Betätigung der RETURN-Taste zu benötigen. Im Beispiel wird das eingegebene Zeichen in der Variable ANS $ gespeichert.

GOSUB 25Ø
Mit dem Befehl verzweigt das Programm auf die angegebene Zeile (25Ø in dem Beispiel). Bei der Ausführung einer RETURN-Anweisung verzweigt das Programm auf die Anweisung, die dem zuletzt ausgeführten GOSUB-Befehl unmittelbar folgt.

GOTO 25Ø
veranlaßt die Verzweigung des Programms auf die angegebene Zeile (25Ø in dem Beispiel).

GR
stellt LR-GRaphikbetrieb (4Ø x 4Ø) für den Bildschirm ein, wobei vier Zeilen für Text am unteren Bildschirmrand gelassen werden. Der Bildschirminhalt wird unter Bildung einer schwarzen Grundfläche gelöscht, der Cursor wird in das nächste Textschreibfeld verschoben und COLOR wird auf Ø (schwarz) gestellt.

HCOLOR=4
stellt die HR-Graphikfarbe auf die durch HCOLOR festgelegte Farbe ein. Die Farbnamen und ihre dazugehörigen Werte sind:

Ø	schwarz 1
1	grün (hängt vom Fernsehgerät ab)
2	blau (hängt vom Fernsehgerät ab)
3	weiß 1

4	schwarz 2
5	(hängt vom Fernsehgerät ab)
6	(hängt vom Fernsehgerät ab)
7	weiß 2

HGR

nur verfügbar in der Firmware-Variante von APPLESOFT. Der Befehl stellt die Bildschirmanzeige auf HR-Graphikbetrieb ein (28Ø x 16Ø), wobei vier Zeilen für Text am unteren Rand des Bildschirms freigelassen werden. Der Bildschirminhalt wird bei gleichzeitiger Bildung einer schwarzen Grundfläche gelöscht,und die Seite 1 des Speichers wird angezeigt. Weder HCOLOR noch der Speicher für den Bildschirminhalt in Textform werden bei der Ausführung von HGR beeinträchtigt. Der Cursor wird nicht in das nächste Textschreibfeld verschoben.

HGR2

stellt auf HR-Graphikbetrieb über den gesamten Bildschirm ein (28Ø x 192). Der Speicher für den Bildschirminhalt in Textform wird nicht beeinflußt.

HIMEN: 16384

setzt die Adresse des höchsten Speicherplatzes, der für ein APPLESOFT-Programm mit seinen Variablen verfügbar ist. Der Befehl wird benutzt, um den Speicherbereich für Daten, HR-Bildschirminhalt oder maschinensprachliche Programme zu schützen. HIMEN: wird nicht durch CLEAR, RUN, DEL, NEW, das Verändern oder Hinzufügen einer Programmzeile oder durch reset auf null gesetzt.

HLIN 1Ø, 2Ø AT 3Ø

wird verwendet, um horizontale Linien im LR-Graphikbetrieb unter Verwendung der zuletzt durch COLOR festgelegten Farbe zu zeichnen. Der Ursprung (x=Ø und y=Ø) für das System ist der Punkt in der äußersten linken oberen Ecke des Bildschirms. In dem Beispiel wird die Linie vom Punkt (x=1Ø, y=3Ø) zum Punkt (x=2Ø, y=3Ø) gezogen.

HOME

versetzt den Cursor in die obere linke Bildschirmposition des Textschreibfeldes und löscht den gesamten Text im Textfeld.

HPLOT 1Ø, 2Ø
HPLOT 3Ø, 4Ø TO 5Ø, 6Ø
HPLOT TO 7Ø, 8Ø

Hiermit werden Punkte und Linien im HR-Graphikbetrieb abgebildet, wobei der zuletzt angegebene Wert von HCOLOR verwendet wird. Der Ursprung befindet sich in der oberen äußersten linken Bildschirmposition (x=Ø, y=Ø). Im ersten Beispiel wird ein HR-Punkt auf x=1Ø, y=2Ø abgebildet. Im zweiten Beispiel wird eine HR-Linie vom Punkt (3Ø, 2Ø) zum Punkt (5Ø,6Ø) gezogen. Das dritte Beispiel bildet eine Linie vom zuletzt gedruckten Punkt bis zum Punkt (7Ø,8Ø) ab, wobei die Farbe des letzten gedruckten Punktes nicht unbedingt die aus dem letzten HCOLOR verwendet wird.

HTAB 23
bewegt den Cursor entweder links oder rechts in die festgelegte Spalte (1 bis 4Ø) auf dem Bildschirm. Im Beispiel wird der Cursor in Spalte 23 gesetzt.

IF AGE<18 THEN A=Ø: B=1: C=2
IF ANS$="YES" THEN GOTO 1ØØ
IF N<MAX THEN 25
IF N<MAX GOTO 25

Wenn der Ausdruck nach IF als wahr angesehen wird (d.h. nicht null), dann wird (werden) die Anweisung(en) nach dem THEN auf der gleichen Zeile ausgeführt. Anderenfalls werden die dem THEN folgenden Befehle ignoriert, und die Programmausführung geht zur nächsten numerierten Programmzeile über. Gruppenausdrücke werden in alphabetischer Reihenfolge bearbeitet. Die Beispiele 2,3 und 4 verhalten sich gleichartig trotz der unterschiedlichen Worte.

INPUT A%
INPUT "TYPE AGE THEN A COMMA THEN NAME": B, C$
Im ersten Beispiel wird durch INPUT ein Fragezeichen abgebildet, was anzeigt, daß der Computer bereit ist, eine vom Benutzer gegebene Zahl entgegenzunehmen. Dieser wird der ganzzahligen Variable A% zugeordnet. Im zweiten Beispiel wird durch INPUT die wahlfreie Gruppe genauso ausgedruckt wie angegeben. Im Anschluß daran muß der Benutzer eine Zahl

(die der reellzahligen Variable B zugeordnet wird), ein Komma und eine Gruppe (die der Gruppenvariable C$ zugeordnet wird) eingegeben. Mehrfacheingaben durch INPUT können durch Kommas oder durch return getrennt werden.

INT(NUM)
gibt die größte ganze Zahl wieder, die kleiner oder gleich dem gegebenen Argument ist. Wenn im Beispiel NUM = 2,389, dann wird 2 ausgegeben. Wenn NUM = -45,123345, dann wird -46 ausgegeben.

INVERSE
stellt den Videobetrieb so ein, daß die Computerausgabe in schwarzen Buchstaben auf weißem Grund gedruckt wird. Mit NORMAL stellen Sie wieder auf weiße Buchstaben auf schwarzem Grund um.

IN # 4
gibt den Eingang (1 bis 7) für das externe Gerät an, über das die nachfolgende Eingabe für den Computer erfolgt. IN Ø stellt wieder auf Eingabe über das Tastenfeld anstelle eines externen Gerätes um.

LEFT$ ("APPLESOFT",5)
gibt die festgelegte Anzahl der äußersten linken Zeichen der Gruppe an. Im Beispiel wird APPLE (die fünf äußersten linken Zeichen) angegeben.

LEN("AN APPLE A DAY")
gibt die Anzahl der Zeichen einer Gruppe wieder, von Ø bis 255, Im Beispiel wird 14 ausgegeben.

LET A = 23,567
A$ = "DELICIOUS"
Der Variablenname links vom Gleichheitszeichen wird dem Wert der Gruppe oder des Ausdrucks rechts vom Gleichheitszeichen zugeordnet. Das LET ist wahlfrei.

LIST
LIST 2ØØ-3ØØØ
LIST 2ØØ, 3ØØØ

Im ersten Beispiel wird das gesamte Programm auf dem Bildschirm angezeigt. Im zweiten Beispiel werden die Zeilen 2ØØ bis 3ØØØ angezeigt. Will man das Programm vom Beginn bis zur Programmzeile 2ØØ auflisten lassen, benutzt man den Befehl LIST -2ØØ. Um das Programm von Zeile 2ØØ bis zum Programmende auflisten zu lassen, verwendet man den Befehl LIST 2ØØ-. Im dritten Beispiel läuft das gleiche wie im zweiten ab. LIST wird durch ctrl C abgebrochen.

LOAD
liest ein APPLESOFT-Programm vom Kassettentonband in den Speicher des Computers ein. Ein Ladesymbol wird nicht gegeben. Der Benutzer muß das Tonband zurückspulen und die Wiedergabetasten des Kassettenrekorders betätigen, bevor er die LOAD-Anweisung gibt. Ein "Piepton" wird hörbar, wenn die Information auf dem Tonband, das geladen wird, erkannt wurde. Wenn das LOAD abgeschlossen ist, ertönt ein zweiter "Piepton", und das Ladesymbol des APPLESOFT erscheint. Nur mit reset kann man das LOAD unterbrechen.

LOG(2)
gibt den natürlichen Logarithmus eines bestimmten arithmetischen Ausdruckes wieder. Im Beispiel wird 0,693147181 angezeigt.

LOMEN: 2Ø6Ø
stellt die Adresse des niedrigsten Speicherplatzes, der für ein BASIC-Programm verfügbar ist, ein. Damit kann man die Variablen vor HR-Graphiken in Computern mit hoher Speicherquantität schützen.

MID$("AN APPLE A DAY",4)
MID$("AN APPLE A DAY",4,9)
gibt die festgelegte Untergruppe wieder. Im ersten Beispiel werden das vierte bis letzte Zeichen der Gruppe wiedergegeben, nämlich APPLE A DAY. Im zweiten Beispiel werden die 9 Zeichen nach dem vierten Zeichen in der Gruppe wiedergegeben, nämlich APPLE A D

NEW
löscht das laufende Programm und alle Variablen.

NEXT
Siehe FOR...TO...STEP

NORMAL

stellt den Videobetrieb auf die normale Schreibart mit weißen Buchstaben auf schwarzem Grund für Ein- und Ausgabe ein.

NOTRACE

schaltet den TRACE-Betrieb ab. Siehe TRACE.

ON ID GOSUB 1ØØ, 2ØØ, 23, 4ØØ5, 5ØØ

führt ein GOSUB zu der Zeilennummer aus, die durch den Wert des arithmetischen Ausdruckes im Anschluß an ON angegeben wird In dem Beispiel wird bei ID=1 GOSUB nach Zeile 1ØØ ausgeführt, bei ID=2 wird GOSUB nach Zeile 2ØØ usw. ausgeführt. Falls der Wert des Ausdruckes gleich Null oder größer als die Zahl der aufgelisteten Wechselzeilennummern ist, setzt sich die Ausführung desProgramms mit der nächsten Anweisung fort

ON ID GOTO 1ØØ, 2ØØ, 23, 4ØØ5, 5ØØ

identisch mit ON ID GOSUB (siehe oben), aber führt eine GOTO-Verzweigung in die Zeilennummer aus, die durch den Wert des arithmetischen Ausdruckes nach ON bestimmt wird.

ONERR GOTO 5ØØ

wird benutzt, um eine Fehlermeldung zu vermeiden, die den Programmablauf bei Auftreten eines Fehlers unterbricht. Bei Ausführung des Befehls wird ein Kennzeichen gesetzt, das einen unbedingten Sprung in die angegebene Zeile (in unserem Beispiel 5ØØ) veranlaßt, wenn es im späteren Verlauf des Programms zu einem Fehler kommt.

PDL(3)

gibt den gegenwärtigen Wert, eine Zahl aus dem Bereich von Ø bis 255, des angegebenen Paddels des Spielreglers wi eder. Spielreglerpaddel sind in der Zahl Ø bis 3 gültig

PEEK(37)

gibt den dezimalen Inhalt eines Bytes wieder auf der benannten dezimalen Adresse (in unserem Beispiel 37).

→ und ← Pfeiltasten

Die Tasten, die mit einem Rechts- oder Linkspfeil beschriftet sind, werden benutzt, um ein APPLESOFT-Programm auch redigieren zu können. Der Rechtspfeil bewegt den Cursor nach

rechts, wobei jedes Zeichen, über das er hinwegstreicht, in den Speicher eingegeben wird, als hätten Sie es selbst eingetastet. Der Linkspfeil bewegt den Cursor nach links, wobei jedesZeichen aus der Programmzeile entfernt wird, die Sie gerade eingeben, unabhängig davon, über welches Zeichen der Cursor hinwegstreicht.

PLOT 1Ø, 2Ø
bildet einen Punkt im LR-Graphikbetrieb an der benannten Stelle ab. Im Beispiel wird ein Punkt mit den Koordinaten x=1Ø und y=2Ø ausgedruckt. Die Farbe des Punktes wird vom zuletzt verarbeiteten COLOR-Befehl vorgegeben. Wenn nicht anders festgelegt, dann wird Ø = schwarz gedruckt.

POKE -163Ø2,Ø
speichert das binäre Äquivalent des zweiten Arguments (Ø in diesem Beispiel) auf der Speicherstelle, deren dezimale Adresse durch das erste Argument (-163Ø2 im Beispiel) gegeben ist.

POP
verursacht ein "Wegnehmen" einer RETURN-Adresse von der Spitze des Stacks der RETURN-Adressen. Das nächste RETURN, das nach einem POP kommt, verursacht dann eine Verzweigung zu einer Anweisung über das vorletzte GOSUB-Kommando hinaus.

POS(Ø)
gibt die gegenwärtige horizontale Position des Läufers wieder. Das ist eine Zahl von Ø (am linken Rand) bis 39 (am rechten Rand). Was Sie in die Klammer setzen ist bedeutungslos. Die Hauptsache ist, daß es durch APPLESOFT verarbeitet werden kann.

PRINT
PRINT A$; "X = "; X
Das erste Beispiel verursacht einen Zeilenvorschub mit Umschaltung zum nächsten Zeilenanfang. Posten, die in einer Liste durch das PRINT-Kommando ausgedruckt werden sollen, sind durch Kommas zu trennen, wenn ein jeder Posten in eine unterschiedliche Tabellenspalte geschrieben werden soll. Sie müssen durch Semikolons getrennt werden, wenn sie gleich nacheinander ohne Leerzeichen dazwischen abgedruckt werden sollen . Wenn A$ "CORE" enthält und X=3 ist, dann wird im zweiten Beispiel COREX = 3 ausgedruckt.

PR # 2

überträgt die Ausgabe zum angegebenen Ausgang des Computers, also zum Ausgang 1 bis 7. Mit PR #Ø wird die Ausgabe über den Bildschirm erfolgen.

READ A, B%, C$

Bei der Ausführung dieses Befehls werden den Variablen in der READ-Anweisung nacheinanderfolgende Werte von Elementen aus der DATA-Anweisung des Programms zugeordnet. Im Beispiel müssen die ersten beiden Elemente in den DATA-Anweisungen Zahlen sein und das dritte Element eine Gruppe (die wiederum auch eine Zahl sein kann) sein. Sie werden jeweils den Variablen A, B% und C$ zugeordnet.

RECALL MX

holt einen reell- oder ganzzahligen Bereich, der auf Kassettentonband gespeichert (STORE) wurde, zurück. Ein Bereich kann auch zurückgeholt werden mit einem Namen, der sich von dem, mit dem der Bereich gespeichert wurde, unterscheidet. Wenn es abgefordert wird, muß MX das Programm dimensioniert (DIM) worden sein. Indizes werden weder bei STORE noch bei RECALL verwendet. Im Beispiel wird der Bereich MX, dessen Elemente MX(Ø), MX(1), ...heißen, abgerufen. Die indexlose Variable MX wird nicht beeinflußt. Es wird kein Ladesymbol angezeigt und auch kein anderes Zeichen gegeben. Sie müssen den Kassettenrecorder auf Wiedergabe eingestellt haben, wenn RECALL ausgeführt wird. "Pieptöne" signalisieren den Anfang und das Ende des aufgenommenen Bereichs. Nur mit reset kann ein RECALL unterbrochen werden.

REM THIS A REMARK

ermöglicht das Einfügen von Bemerkungen in das Programm.

repeat

Wenn Sie die REPEAT-Taste, die mit REPT beschriftet ist, betätigen, während Sie gleichzeitig irgendeine Buchstabentaste drücken, wird das Zeichen wiederholt.

RESTORE

stellt den "DATA-Listenzeiger" wieder auf das erste Element der DATA-Anweisung zurück. Damit wird dafür gesorgt, daß beim Erfolgen der nächsten READ-Anweisung die DATA vom Beginn an noch einmal gelesen werden.

RESUME

Wenn es am Ende des Fehlerbearbeitungsprogramms (siehe ONERR GOTO) steht, löst es die Wiederaufnahme des Programms in der Zeile aus, in deren Anweisung der Fehler aufgetreten war.

RETURN

verzweigt zu der Anweisung, die unmittelbar dem letzten GOSUB folgt.

RIGHT$("SCRAPPLE",5)

gibt die angegebene Zahl der äußersten rechten Zeichen der Gruppe wieder. Im Beispiel wird APPLE (die 5 äußersten rechten Zeichen) ausgegeben.

RND(5)

gibt eine wahlfreie reelle Zahl wieder, für die gilt: $n \geq 0$ und $n < 6$ RND(0) gibt die zuletzt erzeugte wahlfreie Zahl wieder. Jedes negative Argument erzeugt eine besondere wahlfreie Zahl, die jedesmal, wenn sie mit dem gleichen Argument verwendet wird, die gleiche ist. Aufeinanderfolgende RND mit positiven Argumenten werden immer einer besonderen wiederholbaren Reihe folgen. Jedesmal, wenn RND mit einem positiven Argument verwendet wird, wird eine neue wahlfreie Zahl von 0 bis 1 erzeugt, wenn sie nicht Teil der Reihe wahlfreier Zahlen ist, die von einem negativen Argument ausgelöst wurde.

ROT = 16

gibt die Drehung eines von DRAW oder XDRAW abgebildeten Formativs um einen bestimmten Winkel vor. ROT=0 sorgt dafür, daß das Formativ in der Lage gezeichnet wird, wie es festgelegt wurde. ROT=16 verursacht eine Drehung im Uhrzeigersinn von 90° usw. Der Drehprozeß beginnt bei ROT=64 wieder von neuem.

RUN 500

löscht alle Variablen, Zeiger und Stacks und beginnt die Ausführung in der benannten Zeilennummer (hier 500). Wenn keine Zeilennummer angegeben ist, dann setzt die Ausführung des Programms in der Zeile mit der niedrigsten Nummer ein.

SAVE

speichert ein Programm auf Kassettentonband. Es wird kein Ladesymbol oder ein anderes Sig-

nal gegeben. Der Benutzer muß den Kassettenrecorder auf "Aufnahme " einstellten, bevor das SAVE-Kommando gegeben wird. Mit dem SAVE-Kommando wird aber nicht vom Computer geprüft, ob die richtigen Tasten des Kassetterecorders gedrückt sind. "Pieptöne" signalisieren den Beginn und das Ende der Aufnahme.

SCALE=5Ø
stellt die Maßstabgröße für das durch DRAW oder SDRAW abzubildenden Formativ ein. SCALE=1 realisiert eine 1:1-Reproduktion der Formativdefinition. SCALE=255 führt dazu, daß jeder Vektor 255 mal vergrößert abgebildet wird. Beachte: SCALE=Ø bedeutet Maximalgröße und nicht Einzelpunktgröße.

SCRN(1Ø, 2Ø)
im LR-Graphikbetrieb gibt es den Farbcode eines bestimmten Punktes wieder. In unserem Beispiel wird die Farbe des Punktes mit den Koordinaten x=1Ø und y=2Ø ausgegeben.

SGN(NUM)
gibt -1 aus, wenn das Argument negativ ist, Ø, wenn es null ist, und +1, wenn es positiv ist.

SHLOAD
lädt eine Formativtabelle vom Kassettentonband. Die Formativtabelle wird unmittelbar unter HIMEM: geladen, und dann wird HIMEM: unmittelbar unter die Formativtabelle gesetzt, um sie zu schützen.

SIN(2)
gibt den Sinus des Arguments wieder, welcher im Bogenmaß angegeben wird. In unserem Beispiel wird Ø.9Ø9297427 ausgegeben.

SPC(8)
muß in einer PRINT-Anweisung benutzt werden. Führt die benannte Anzahl von Leerzeichen (8 in unserem Beispiel) zwischen das zuletzt gedruckte und das als nächstes zu druckende Zeichen ein, wenn dem SPC-Befehl ein Semikolon folgt oder vorausgeht.

SPEED = 5Ø
setzt die Rate fest, mit der die Zeichen zum Bildschirm oder zu anderen Ein- und Ausgabege-

räten geschickt werden. Die Mindestgeschwindigkeit ist Ø, die Höchstgeschwindigkeit 255.

SQR(2)

gibt die positive Quadratwurzel des Arguments wieder. In unserem Beispiel wird 1,41421366 ausgegeben. SQR wird schneller ausgeführt als ^ .5.

STOP

läßt den Programmablauf stoppen und anzeigen, in welcher Zeile das STOP enthalten ist. Dem Benutzer wird die Steuerung des Computers übergeben.

STORE MX

zeichnet einen ganzzahligen oder reellzahligen Bereich auf Tonband auf. Es wird keine Bereitschaftsmeldung oder ein anderes Signal gegeben. Der Nutzer muß selbst den Recorder auf "Aufnahme" einstellen, wenn das STORE-Kommando ausgeführt wird. "Pieptöne" kündigen den Anfang und das Ende der Aufzeichnung an. Die Indizes der Bereiche werden nicht angegeben, wenn STORE verwendet wird. Im Beispiel werden die Elemente MX(Ø), MX(1), MX(2)... auf Tonband gespeichert. Die Variable MX wird nicht beeinflußt. Siehe RECALL.

STR$(12.45)

gibt eine Gruppe wieder, die den Wert des Arguments darstellt. Im Beispiel wird die Gruppe "12.45" ausgegeben.

TAB(23)

muß in einer PRINT-Anweisung benutzt werden. Das Argument muß zwischen Ø und 255 liegen und in Klammern stehen. Wenn das Argument, welches zwischen 1 und 255 liegt, größer als der Wert der laufenden Cursor-Position ist, dann wird durch TAB der Cursor auf die benannte PRINT-Position, gerechnet vom linken Rand der laufenden Zeile an, in der sich der Cursor befindet, verschoben. Sollte das Argument kleiner als der Wert der laufenden Cursor-Position sein, dann wird er nicht bewegt. TAB(Ø) versetzt den Cursor auf Position 256.

TAN(2)

gibt den Tangens des Arguments im Bogenmaß wieder. In unserem Beispiel wird -2,285Ø3987 ausgegeben.

TEXT

stellt den Bildschirm auf die gewöhnliche Betriebsart ein, in der keine graphischen Darstellungen erfolgen. Die Anzeige verfügt dabei über 24 Zeilen zu je 4Ø Zeichen. Mit diesem Kommando wird auch das Textschreibfeld auf die Größe des gesamten Bildschirms gesetzt.

TRACE

sorgt dafür, daß jede Zeilennummer einer jeden Anweisung auf dem Bildschirm angezeigt wird, wenn sie ausgeführt wird. TRACE wird durch RUN, CLEAR, NEW, DEL oder reset abgestellt. NOTRACE stellt TRACE auch ab.

USR(3)

Diese Funktion überträgt ihre Anweisung in ein maschinensprachliches Unterprogramm. Das Argument wird berechnet und in den Gleitkomma-Akkumulator (Speicherstellen $9D bis $A3) gegeben. Dann wird ein JSR zur Speicherstelle $ØA ausgeführt. Von Speicherstelle $ØA bis $ØC muß ein JMP zur Anfangsspeicherstelle des maschinensprachlichen Unterprogramms enthalten sein. Der für die Funktion wiedergegebene Wert wird in den Gleitkomma-Akkumulator gegeben. Um zum APPLESOFT zurückzukehren, führen Sie ein RTS aus.

VAL("-3.7E4A5PLE")

versucht eine Gruppe zu interpretieren, und zwar bis zum ersten nichtnumerischen Zeichen, wie beispielsweise eine reelle oder ganze Zahl, und gibt den Wert der Zahl wieder. Wenn keine Zahl vor dem ersten nichtnumerischen Zeichen vorkommt, dann wird die Ø ausgegeben. Im Beispiel wird -37ØØØ ausgegeben.

VLIN 1Ø, 2Ø AT 3Ø

zieht im LR-Graphikbetrieb eine vertikale Linie in der vom letzten COLOR-Befehl benannten Farbe. Die Linie wird in der Spalte gezogen, die vom dritten Argument angegeben wird. Im Beispiel wird die Linie von y=1Ø nach y=2Ø bei x=3Ø gezogen.

VTAB(15)

versetzt den Cursor in die Zeile, die von dem Argument angegeben wird. Die oberste Zeile des Bildschirms trägt die Nr. 1, die untere die Nr. 24. VTAB bewegt den Cursor nach oben oder unten bzw. nach rechts oder links.

WAIT 16ØØØ, 255

WAIT 16ØØØ, 255, Ø

ermöglicht das Einfügen einer bedingten Pause in das Programm. Das erste Argument ist eine dezimale Adresse eines Speicherplatzes, der getestet wird, um herauszufinden, ob ein gewisses Bit im hohen Zustand (1 oder ein) und ein gewisses anderes Bit im niedrigen Zustand (Ø oder aus) ist. Jedes Bit in dem binären Äquivalent des dezimalen zweiten Arguments gibt an, ob Sie an dem entsprechenden Bit auf der Speicherstelle interessiert sind oder nicht. 1 bedeutet ja; Ø bedeutet nein. Jedes Bit in dem binären Äquivalent des dezimalen dritten Arguments gibt an, in welchem Zustand Sie das entsprechende Bit auf dem Speicherplatz erwarten. 1 bedeutet, das Bit muß im niedrigen Zustand sein, Ø bedeutet, das Bit muß im hohen Zustand sein. Wenn kein drittes Argument vorhanden ist, dann wird Ø angenommen. Wenn ein beliebiges Bit, das durch ein 1-Bit im zweiten Argument angegeben ist, mit dem Zustand für das Bit, das durch das entsprechende Bit im dritten Argument angegeben ist, zusammenpaßt, dann ist das WAIT vollendet.

XDRAW 3 AT 18Ø, 12Ø

zeichnet die Formativdefinition Nr. 3 aus einer vorher geladenen Formativtabelle im HR-Graphikbetrieb nach. Es wird im Punkt x=18Ø, y=12Ø begonnen. Jeder Punkt,der abgebildet wird, erscheint in der Komplementärfarbe der Farbe, die sich an dem Punkt bereits befindet. Es ermöglichst eine sehr bequeme Art des Löschens. Wenn ein Formativ mit XDRAW gezeichnet wird, und dann ein zweites Mal XDRAW eingegeben wird, dann wird das Formativ gelöscht, ohne daß der Untergrund gelöscht wird.

